FDK Programmer's Reference

VERSION 11

Frame Developer's Kit,
November 2012

Adobe Systems Incorporated
Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000

© 2012 Adobe Systems Incorporated and its licensors. All rights reserved.

Adobe® FrameMaker® 11 for Windows® Frame Developer's Kit for Windows

This manual is protected under copyright law, furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this manual.

This manual is licensed for use under the terms of the Creative Commons Attribution Non-Commercial 3.0 License. This License allows users to copy, distribute, and transmit the manual for noncommercial purposes only so long as (1) proper attribution to Adobe is given as the owner of the manual; and (2) any reuse or distribution of the manual contains a notice that use of the manual is governed by these terms. The best way to provide notice is to include the following link. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/us/

Adobe, the Adobe logo, Adobe AIR, Adobe Captivate, Adobe Type Manager, Acrobat, AIR, Creative Suite, Distiller, Flash, FrameMaker, Illustrator, PageMaker, Photoshop, PostScript, Reader, RoboHelp, and RoboScreenCapture are trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Microsoft, Windows, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Solaris is a trademark or registered trademark of Oracle and/or its affiliate. SVG is a trademark of the World Wide Web Consortium; marks of the W3C are registered and held by its host institutions MIT, INRIA, and Keio.

Updated Information/Additional Third Party Code Information available at http://www.adobe.com/go/thirdparty.

Portions include software under the following terms:

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

This Program was written with MacApp®: ©1985-1988 Apple Computer, Inc. APPLE COMPUTER, INC. MAKES NO WARRANTIES WHATSOEVER, EITHER EXPRESS OR IMPLIED, REGARDING THIS PRODUCT, INCLUDING WARRANTIES WITH RESPECT TO ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. The MacApp software is proprietary to Apple Computer, Inc. and is licensed to Adobe for distribution only fors use in combination with Adobe FrameMaker.

Portions copyright 1984-1998 Faircom Corporation. "FairCom" and "c-tree Plus" are trademarks of Faircom Corporation and are registered in the United States and other countries. All Rights Reserved.

MPEG Layer-3 audio compression technology licensed by Fraunhofer IIS and THOMSON multimedia (http://www.iis.fhg.de/amm).

ImageStream Graphics Filters and ImageStream are registered trademarks of Inso Corporation.

Portions utilize code licensed from Nellymoser (www.nellymoser.com)

Adobe Flash Player 10 video compression and decompression is powered by On2 TrueMotion video technology. © 1992-2005 On2 Technologies, Inc. All Rights Reserved. http://www.on2.com.

Certain trademarks are owned by The Proximity Division of Franklin Electronic Publishers, Inc., and are used by permission. Merriam-Webster is a trademark of Merriam-Webster, Inc.

This product contains either BSAFE and/or TIPEM software by RSA Data Security, Inc.

Sorenson Spark™ video compression and decompression technology licensed from Sorenson Media, Inc.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. government end users. The software and documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

The latest version of this document is available online at

http://www.adobe.com/devnet/framemaker.html

Contents

1 What's New in FDK 11 17

Key catalogs for key-based (indirect) referencing 17
New APIs for key catalog functionality 18
New FDK APIs to update DITA references 18
Support for object styles 19
Support for multiple views 19
Support for hotspots in objects 19
Interactive multimedia links for 3D objects 20
Support for referring to content using line numbers 21
Support for guided structured authoring with banner text 21
New APIs, notifications, error codes, objects, and properties 21
New APIs 21
New notifications 22
New error codes 22
New objects 23
New properties 23
CMS Connector Framework 26

2 Function Summary 27

Client-defined callback functions 28
Books 28
Characters 30
Clipboard 31
Commands and menus 31
Data structures: copying 33
Debugging 33
DITA references: updating 35
Dialog boxes: predefined 35
Dialog boxes: client-defined 35
Documents: comparing 37
Documents: file operations 37
Documents: formatting 38
Documents: updating 38
F-codes 39
Files, directories, and filepaths 39
Fonts 41
Graphic insets 41
Hash tables 42
Hypertext 42

I/O 42
Insets 43
Key Catalogs 44
Memory: allocating and deallocating structures 44
Memory: manipulating with handles 46
Memory: manipulating with pointers 46
Menus 47
Metrics 48
Maker Interchange Format (MIF) 48
Objects: creating and deleting 50
Objects: getting IDs 52
Printing 52
Properties 52
Selection 54
Sessions 54
Sleep 54
String lists 54
Strings: allocating, copying, and deallocating 56
Strings: comparing and parsing 56
Strings: concatenating 58
Strings: miscellaneous 58
Strings: encoded 59
Structure: manipulating elements 59
Structure: manipulating element definitions and format rules 61
Tables 61
Text 63
Text: find and replace 63
Text insets 64
Unicode 64
Undo/Redo 64
Utility 66

3 FDK Function Reference 67

F_Alloc() 68
F_AllocHandle() 69
F_ApiAddCols() 70
F_ApiAddCommandToMenu() 72
F_ApiAddMenuToMenu() 74
F_ApiAddRows() 78
F_ApiAddText() 80
F_ApiAlert() 82
F_ApiAlive() 85
F_ApiAllocatePropVals() 85
F_ApiAllocateTextItems() 86
F_ApiApplyAttributeExpression() 88
F_ApiApplyAttributeExpressionAsCondition() 89
F_ApiApplyPageLayout() 90
F_ApiBailOut() 92

F_ApiCallClient() 93
F_ApiCenterOnText() 94
F_ApiCheckStatus() 96
F_ApiChooseFile() 97
F_ApiClear() 100
F_ApiClearAllChangebars() 102
F_ApiClientDir() 103
F_ApiClientName() 104
F_ApiClientPath() 106
F_ApiClose() 107
F_ApiCombinedFamilyFonts() 109
F_ApiCommand() 111
F_ApiCompare() 112
F_ApiConnectToSession() 115
F_ApiCopy() 115
F_ApiCopy*StructureType*() 118
F_ApiCustomDoc() 119
F_ApiCut() 123
F_ApiDeallocate*StructureType*() 125
F_ApiDefineAndAddCommand() 126
F_ApiDefineAndAddCommandEx() 129
F_ApiDefineAndAddMenu() 134
F_ApiDefineCommand() 137
F_ApiDefineCommandEx() 139
F_ApiDefineMenu() 141
F_ApiDelete() 143
F_ApiDeleteAllKeyDefinitions() 144
F_ApiDeleteCols() 145
F_ApiDeletePropByName() 147
F_ApiDeleteRows() 148
F_ApiDeleteText() 151
F_ApiDeleteTextInsetContents() 152
F_ApiDialogEvent() 155
F_ApiDialogItemId() 158
F_ApiDisconnectFromSession() 159
F_ApiEnableUnicode() 162
F_ApiErr() 164
F_ApiFamilyFonts() 165
F_ApiFcodes() 167
F_ApiFind() 168
F_ApiGetAllKeyDefinitions() 175
F_ApiGetAllKeys() 185
F_ApiGetEncodingForFamily() 187
F_ApiGetEncodingForFont() 189
F_ApiGetId() 191
F_ApiGetImportDefaultParams() 193
F_ApiGetInt() 201
F_ApiGetIntByName() 203

F_ApiGetInts() 204
F_ApiGetKeyCatalog() 206
F_ApiGetKeyDefinition() 207
F_ApiGetMetric() 209
F_ApiGetMetricByName() 210
F_ApiGetMetrics() 212
F_ApiGetNamedObject() 214
F_ApiGetNewXMLDefaultParams() 215
F_ApiGetObjectType() 218
F_ApiGetOpenDefaultParams() 219
F_ApiGetPoints() 228
F_ApiGetPropIndex() 230
F_ApiGetProps() 231
F_ApiGetPropVal() 232
F_ApiGetSaveDefaultParams() 234
F_ApiGetString() 241
F_ApiGetStrings() 242
F_ApiGetSupportedEncodings() 244
F_ApiGetTabs() 245
F_ApiGetText() 249
F_ApiGetText2() 256
F_ApiGetTextForRange() 260
F_ApiGetTextForRange2() 261
F_ApiGetTextLoc() 264
F_ApiGetTextProps() 265
F_ApiGetTextPropVal() 267
F_ApiGetTextRange() 269
F_ApiGetTextVal() 271
F_ApiGetUBytesByName() 273
F_ApiGetUniqueObject() 275
F_ApiGetUpdateBookDefaultParams() 278
F_ApiGetVal() 282
F_ApiHypertextCommand() 285
F_ApiImport() 286
F_ApiInitialize() 293
F_ApiIsEncodingSupported() 295
F_ApiLoadMenuCustomizationFile() 296
F_ApiMakeTblSelection() 297
F_ApiMenuItemInMenu() 298
F_ApiMessage() 303
F_ApiModalDialog() 303
F_ApiModelessDialog() 305
F_ApiMoveComponent() 306
F_ApiNetLibSetAuthFunction() 307
F_ApiNewAnchoredFormattedObject() 310
F_ApiNewAnchoredObject() 312
F_ApiNewBookComponentOfTypeInHierarchy() 316
F_ApiNewGraphicObject() 322

F_ApiNewKeyCatalog() ... 324
F_ApiNewKeyDefinition() ... 325
F_ApiNewNamedObject() ... 327
F_ApiNewSeriesObject() ... 329
F_ApiNewTable() ... 336
F_ApiNewXML() ... 338
F_ApiNotification() ... 339
F_ApiNotify() ... 349
F_ApiObjectValid() ... 354
F_ApiOpen() ... 355
F_ApiOpenResource() ... 363
F_ApiPaste() ... 364
F_ApiPopClipboard() ... 366
F_ApiPrintFAErrno() ... 367
F_ApiPrintImportStatus() ... 369
F_ApiPrintOpenStatus() ... 370
F_ApiPrintPropVal() ... 370
F_ApiPrintPropVals() ... 371
F_ApiPrintSaveStatus() ... 372
F_ApiPrintTextItem() ... 373
F_ApiPrintTextItems() ... 374
F_ApiPrintUpdateBookStatus() ... 375
F_ApiProgressBarEx() ... 376
F_ApiPromptInt() ... 379
F_ApiPromptMetric() ... 381
F_ApiPromptString() ... 382
F_ApiPushClipboard() ... 384
F_ApiQuickSelect() ... 385
F_ApiRedisplay() ... 386
F_ApiReformat() ... 388
F_ApiRehyphenate() ... 389
F_ApiResetEqnSettings() ... 390
F_ApiResetReferenceFrames() ... 391
F_ApiRestartPgfNumbering() ... 392
F_ApiReturnValue() ... 393
F_ApiSave() ... 399
F_ApiScrollBox() ... 402
F_ApiScrollToText() ... 404
F_ApiService() ... 405
F_ApiSetAttributeSimple() ... 411
F_ApiSetClientDir() ... 412
F_ApiSetId() ... 414
F_ApiSetInt() ... 417
F_ApiSetIntByName() ... 419
F_ApiSetInts() ... 421
F_ApiSetMetric() ... 423
F_ApiSetMetricByName() ... 425
F_ApiSetMetrics() ... 426

F_ApiSetPoints() 429
F_ApiSetProps() 432
F_ApiSetPropVal() 434
F_ApiSetString() 436
F_ApiSetStrings() 438
F_ApiSetTabs() 441
F_ApiSetTextLoc() 444
F_ApiSetTextProps() 445
F_ApiSetTextPropVal() 447
F_ApiSetTextRange() 450
F_ApiSetTextVal() 452
F_ApiSetUBytesByName() 454
F_ApiShutDown() 455
F_ApiSilentPrintDoc() 456
F_ApiSimpleGenerate() 459
F_ApiSimpleImportFormats() 462
F_ApiSimpleNewDoc() 464
F_ApiSimpleOpen() 465
F_ApiSimpleSave() 466
F_ApiSleep() 468
F_ApiStraddleCells() 470
F_ApiStringLen() 471
F_ApiUndoCancel() 474
F_ApiUndoEndCheckPoint() 475
F_ApiUndoStartCheckPoint() 476
.F_ApiUnStraddleCells() 477
F_ApiUpdateBook() 480
F_ApiUpdateDITAReference() 483
F_ApiUpdateDITAReferences() 485
F_ApiUpdateKeyDefinition() 487
F_ApiUpdateTextInset() 488
F_ApiUpdateVariables() 489
F_ApiUpdateXRef() 490
F_ApiUpdateXRefs() 492
F_ApiUserCancel() 494
F_ApiUSleep() 494
F_ApiWinConnectSession() 495
F_ApiWinInstallDefaultMessageFilter() 496
F_Assert() 498
F_Calloc() 499
F_ChannelAppend() 500
F_ChannelClose() 501
F_ChannelCloseTmp() 502
F_ChannelEof() 502
F_ChannelFlush() 503
F_ChannelMakeTmp() 503
F_ChannelOpen() 504
F_ChannelPeek() 506

F_ChannelRead() 506
F_ChannelSeek() 508
F_ChannelSize() 509
F_ChannelTell() 510
F_ChannelWrite() 511
F_CharIsAlphabetic() 512
F_CharIsAlphaUTF8() 513
F_CharIsAlphaNumeric() 514
F_CharIsAlphaNumericUTF8() 515
F_CharIsControl() 515
F_CharIsControlUTF8() 516
F_CharIsDoubleByte() 517
F_CharIsDoubleByteFirst() 518
F_CharIsDoubleByteSecond() 520
F_CharIsEol() 521
F_CharIsEolUTF8() 522
F_CharIsHexadecimal() 523
F_CharIsHexadecimalUTF8() 524
F_CharIsLower() 525
F_CharIsLowerUTF8() 525
F_CharIsNumeric() 526
F_CharIsNumericUTF8() 527
F_CharIsUpper() 528
F_CharIsUpperUTF8() 528
F_CharToLower() 529
F_CharToLowerUTF8() 530
F_CharToUpper() 531
F_CharToUpperUTF8() 532
F_CharUTF16ToUTF8() 533
F_CharUTF32ToUTF8() 534
F_CharUTF8ToUTF16() 535
F_CharUTF8ToUTF32() 536
F_ClearHandle() 537
F_ClearPtr() 537
F_CopyPtr() 538
F_DeleteFile() 539
F_DigitValue() 540
F_DuplicateHandle() 541
F_DuplicatePtr() 542
F_Exit() 543
F_FdeEncodingsInitialized() 544
F_FdeInit() 544
F_FdeInitFontEncs() 545
F_FilePathBaseName() 547
F_FilePathCloseDir() 548
F_FilePathCopy() 548
F_FilePathFree() 549
F_FilePathGetNext() 550

F_FilePathOpenDir() 551
F_FilePathParent() 552
F_FilePathProperty() 553
F_FilePathToPathName() 554
F_FontEncId() 556
F_FontEncName() 557
F_FontEncToTextEnc() 558
F_Free() 559
F_FreeHandle() 560
F_GetFilePath() 561
F_GetICUDataDir() 561
F_GetHandleSize() 562
F_HandleEqual() 563
F_HashCreate() 564
F_HashDestroy() 568
F_HashEnumerate() 568
F_HashGet() 570
F_HashRemove() 570
F_HashReportOnData() 571
F_HashSet() 573
F_IsValidUTF8() 573
F_LanguageNumber() 574
F_LanguageString() 576
F_LockHandle() 577
F_MakeDir() 579
F_MetricApproxEqual() 579
F_MetricConstrainAngle() 580
F_MetricDiv() 582
F_MetricFloat() 582
F_MetricFractMul() 583
F_MetricMake() 584
F_MetricMul() 585
F_MetricNormalizeAngle() 585
F_MetricSqrt() 587
F_MetricSquare() 587
F_MetricToFloat() 588
F_MifBegin() 589
F_MifComment() 589
F_MifDecimal() 590
F_MifEnd() 591
F_MifGetIndent() 593
F_MifIndent() 593
F_MifIndentDec() 594
F_MifIndentInc() 595
F_MifInteger() 595
F_MifNewLine() 596
F_MifSetIndent() 597
F_MifSetOutputChannel() 598

F_MifSpace() ... 599
F_MifTab() ... 599
F_MifText() ... 600
F_MifTextString() ... 601
Simple MIF library ... 601
F_PathNameToFilePath() ... 606
F_PathNameType() ... 608
F_Printf() ... 608
F_Progress() ... 610
F_PtrEqual() ... 611
F_ReadBytes() ... 613
F_ReadLongs() ... 614
F_ReadShorts() ... 615
F_Realloc() ... 616
F_ReallocHandle() ... 618
F_RenameFile() ... 620
F_ResetByteOrder() ... 620
F_ResetDirHandle() ... 622
F_Scanf() ... 623
F_SetAssert() ... 624
F_SetByteOrder() ... 625
F_SetDSExit() ... 625
F_SetICUDataDir() ... 626
F_Sprintf() ... 628
F_Sscanf() ... 630
F_StrAlphaToInt() ... 631
F_StrAlphaToIntUnicode() ... 631
F_StrAlphaToReal() ... 633
F_StrAlphaToRealUnicode() ... 633
F_StrBrk() ... 635
F_StrBrkUTF8 () ... 635
F_StrCat() ... 637
F_StrCatCharN() ... 638
F_StrCatDblCharNEnc() ... 639
F_StrCatIntN() ... 640
F_StrCatN() ... 642
F_StrCatNEnc() ... 643
F_StrCatUTF8CharNByte () ... 644
F_StrChr() ... 645
F_StrChrEnc() ... 646
F_StrChrUTF8() ... 648
F_StrCmp() ... 649
F_StrCmpN() ... 650
F_StrCmpNEnc() ... 650
F_StrICmpNUTF8Char ... 651
F_StrCmpUTF8() ... 652
F_StrCmpUTF8Locale() ... 654
F_StrICmpUTF8Locale() ... 655

F_StrConvertEnc(), F_StrConvertEnc_IgnoreControlChars(), F_StrConvertEnc_Convert-ControlChars() 655
F_StrCopyString() 658
F_StrCpy() 659
F_StrCpyN() 660
F_StrCpyNEnc() 661
F_StrCpyNUTF8Char () 662
F_StrEqual() 663
F_StrEqualN() 664
F_StrICmp() 664
F_StrICmpEnc() 665
F_StrICmpN() 666
F_StrICmpNEnc() 667
F_StrIEqual() 668
F_StrIEqualEnc() 669
F_StrIEqualN() 670
F_StrIEqualNEnc() 671
F_StrIEqualNUTF8Char () 672
F_StrIPrefixEnc() 673
F_StrIsEmpty() 674
F_StrISuffixEnc() 675
F_StrLen() 676
F_StrLenEnc() 676
F_StrLenUTF16() 677
F_StrListAppend() 678
F_StrListCat() 679
F_StrListCopy() 679
F_StrListCopyList() 681
F_StrListFirst() 682
F_StrListFree() 683
F_StrListGet() 684
F_StrListIIndex() 684
F_StrListIndex() 686
F_StrListInsert() 687
F_StrListLast() 687
F_StrListLen() 688
F_StrListNew() 689
F_StrListRemove() 690
F_StrListSetString() 690
F_StrListSort() 691
F_StrNew() 693
F_StrPrefix() 694
F_StrPrefixN() 695
F_StrRChr() 695
F_StrRChrEnc() 696
F_StrReverse() 698
F_StrReverseUTF8Char () 698
F_StrStrip() 701

F_StrStripLeadingSpaces() ... 701
F_StrStripTrailingSpaces() ... 702
F_StrStripUTF8Chars () ... 703
F_StrStripUTF8String () ... 704
F_StrStripUTF8Strings () ... 705
F_StrSubString() ... 706
F_StrSuffix() ... 707
F_StrTok() ... 708
F_StrTokUTF8 () ... 709
F_StrTrunc() ... 710
F_StrTruncEnc() ... 711
F_TextEncToFontEnc() ... 712
F_UnlockHandle() ... 714
F_UTF16CharSize() ... 714
F_UTF16NextChar() ... 715
F_UTF8CharSize() ... 716
F_UTF8NextChar() ... 717
F_Warning() ... 719
F_WriteBytes() ... 719
F_WriteLongs() ... 720
F_WriteShorts() ... 721

4 Object Reference ... 723

Books ... 723
FO_Book ... 723
FO_BookComponent ... 739
Character formats ... 750
FO_CharFmt ... 750
Colors ... 755
FO_Color ... 755
Columns ... 757
Combined font definitions ... 758
FO_CombinedFontDefn ... 759
Commands, menus, menu items, and menu item separators ... 760
Common command, menu, and menu item separator properties ... 760
FO_Command ... 761
FO_MenuItemSeparator ... 766
FO_Menu ... 766
Conditions ... 767
FO_AttrCondExpr ... 767
FO_CondFmt ... 767
Cross-references ... 769
FO_XRef ... 769
FO_XRefFmt ... 770
Dialog boxes ... 770
FO_DialogResource ... 771
FO_DlgBox ... 774
FO_DlgButton ... 775

FO_DlgCheckBox ... 775
FO_DlgEditBox ... 776
FO_DlgImage ... 776
FO_DlgLabel ... 776
FO_DlgPopUp ... 777
FO_DlgRadioButton ... 777
FO_DlgScrollBar ... 778
FO_DlgScrollBox ... 779
FO_DlgTriBox ... 780
Documents ... 780
FO_Doc ... 780
Elements ... 829
Flows ... 829
FO_Flow ... 829
Footnotes ... 830
FO_Fn ... 830
Format change lists ... 831
FO_FmtChangeList ... 831
Format rules ... 841
FO_FmtRule ... 841
FO_FmtRuleClause ... 842
Frames ... 844
Graphics ... 844
FO_AFrame ... 849
FO_Arc ... 851
FO_Ellipse ... 852
FO_Group ... 852
FO_Inset ... 852
FO_Iterator ... 858
FO_KeyCatalog ... 859
FO_Line ... 860
FO_Math ... 860
FO_Polygon ... 861
FO_Polyline ... 861
FO_Rectangle ... 862
FO_RoundRect ... 862
FO_TextFrame ... 863
FO_TextLine ... 865
FO_UnanchoredFrame ... 867
FO_Graphicsfmt ... 868
Insets ... 872
Markers ... 872
FO_Marker ... 872
Marker types ... 873
FO_MarkerType ... 873
Menus ... 873
Menu items ... 874
Pages ... 874

FO_BodyPage ... 874
FO_HiddenPage ... 875
FO_MasterPage ... 875
FO_RefPage ... 876
Paragraphs ... 877
FO_Pgf ... 877
FO_PgfFmt ... 887
Rubi composites ... 898
FO_Rubi ... 898
Table ruling formats ... 899
FO_RulingFmt ... 899
Separators ... 899
Session ... 899
FO_Session ... 900
Structural elements ... 909
FO_Element ... 909
FO_ElementDef ... 917
Tables ... 920
FO_Cell ... 921
FO_Row ... 924
FO_Tbl ... 926
Table formats ... 933
FO_TblFmt ... 933
Text columns ... 937
Text frames ... 937
Text insets ... 937
Common text inset properties ... 938
FO_TiApiClient properties ... 942
FO_TiFlow properties ... 942
FO_TiText properties ... 943
FO_TiTextTable properties ... 944
Text properties ... 944
Variables ... 949
FO_Var ... 949
FO_VarFmt ... 949

5 Data Types and Structures Reference ... 953

Data types ... 953
MetricT values ... 954
Data structures ... 955
ChannelT ... 955
DirHandleT ... 956
FilePathT ... 956
HandleT ... 956
HashT ... 956
StringListT ... 956
F_AttributeDefsT ... 956
F_AttributeDefT ... 957

F_AttributesT 958
F_AttributeT 958
F_CombinedFontsT 958
F_CombinedFontT 959
F_CompareRetT 959
F_ElementLocT 959
F_ElementRangeT 960
F_FontEncT 960
F_FontsT 960
F_FontT 961
F_ElementCatalogEntryT 961
F_ElementCatalogEntriesT 962
F_IntsT 962
F_MetricsT 962
F_PointT 962
F_PointsT 962
F_PropIdentT 963
F_PropValT 963
F_PropValsT 963
F_StringsT 963
F_TabT 964
F_TabsT 964
F_TextItemT 964
F_TextItemsT 968
F_TextLocT 968
F_TextRangeT 968
F_TypedValT 969
F_UBytesT 970
F_UIntsT 970

6 Error Codes 971

7 Calling Clients Shipped with FrameMaker 977
Calls to work with structured documents 977
Calls to Sort Tables 984
Calls for WebDAV workgroup management 986
Call to work with book error logs 990

8 CMS Connector FrameWork 993
CMS API Data Structures and Enum Constants 993
F_CMSResultT 993
F_CMSOpResultT 993
F_CMSPropertyT 994
F_CMSItemPropertyT 994
F_CMSItemTypeValueT 995
F_CMSItemFileTypeValueT 996
F_CMSPropertiesT 996

F_CMSMenuItemT 997
F_CMSMenuTypeT 997
F_CMSVersioningStrategyT 998
F_CMSDeleteParamT 998
F_CMSInfoT 999
F_CMSInfosT 999
Error Codes 1000
CMS API Functions 1001
F_ApiAllocateCMSInfos 1001
F_ ApiDeallocateCMSInfos 1002
F_ ApiAllocateCMSProperties 1002
F_ApiDeallocateCMSProperties 1003
F_ApiCMSCommand 1004
F_CMSCommandsT 1005
F_CMSCommandArgsIdT 1007
F_ApiCMSRegister 1010
F_ApiCMSCreateObject 1011
F_ApiCMSEnableCommand 1012
F_ApiCMSDisableCommand 1013
F_ApiCMSAddMenuEntry 1014
F_ApiCMSGetProperties 1015
F_ApiCMSGetProperty 1016
F_ApiCMSSetProperties 1017
F_ApiCMSSetProperty 1019
F_ApiCMSConfigLoginUI 1021
F_ApiCMSShowCheckoutUI 1022
F_CMSCustomizeCheckoutUI 1022
F_ApiCMSShowCheckinUI 1023
F_ApiCMSShowCancelCheckoutUI 1024
F_ApiCMSShowDeleteUI 1025
F_ApiCMSShowCommonListUI 1026
F_ApiCMSShowPropertyUI 1028
F_ApiCMSShowBrowseRepositoryUI 1029
F_ApiCMSGetCmsIdFromName 1030
F_ApiCMSGetCMSInfo 1031
F_ApiCMSGetCmsIdFromSession 1032
F_ApiCMSGetCmsInfoList 1032

9 APIs to automate the CMS connectors functionality 1035
F_ApiCMSLogin 1035
F_ApiCMSLogout 1036
F_ApiCMSCheckout 1037
F_ApiCMSCheckin 1038
F_ApiCMSCancelCheckout 1039
F_ApiCMSDelete 1040
F_ApiCMSOpenFile 1041
F_ApiCMSUploadObject 1042
F_ApiCMSDownloadObject 1043

F_ApiGetCMSObjectFromPath..... 1044

1

What's New in FDK 11

This chapter provides an overview of the new features in FDK 11.

Key catalogs for key-based (indirect) referencing

FDK 11 supports key catalogs for indirect referencing. You can define a key for a file or resource in your project; updating the key will update the references to the file or resource everywhere.

A key catalog is a collection of all the keys in your project—you can manage all the keys from the key catalog. A key catalog is represented by the `FO_KeyCatalog` object.

The key mapping consists of:

- Key name
- Referenced file
- Type of the source file containing the key definition
- Source file containing the key definition
- Information for using the key as a text variable

The key catalog consists of:

- A unique key catalog tag
- A hash-based list of key mappings
- Count of key mappings (including duplicate ones)
- Type of the source file containing the key catalog (KEYDEFSRC_NONE, KEYDEFSRC_DITAMAP)
- Source of key catalog. (DITA Map or other document holding the key catalog)
- Name of the client owning the key catalog.
- Appropriate Flags

New APIs for key catalog functionality

The following new APIs support the key catalog functionality

- `F_ApiNewKeyCatalog()`

  Creates a new key catalog with the specified tag.

- `F_ApiGetKeyCatalog()`

  Finds a key catalog with the specified tag.

- `F_ApiNewKeyDefinition()`

  Adds a new key definition to the specified key catalog.

- `F_ApiUpdateKeyDefinition()`

  Updates the specified key definition field for the specified key in the specified key catalog.

- `F_ApiGetKeyDefinition()`

  Gets the specified key definition field for the specified key from the specified key catalog.

- `F_ApiGetAllKeyDefinitions()`

  Gets all the key definitions from the specified key catalog.

- `F_ApiDeleteAllKeyDefinitions()`

  Deletes all the key definitions in the specified key catalog.

- `F_ApiGetAllKeys()`

  Gets all the key tags from the specified key catalog.

New FDK APIs to update DITA references

The following new APIs have been provided to update DITA references:

- `F_ApiUpdateDITAReference()`

  Updates the DITA object represented by the specified element.

- `F_ApiUpdateDITAReferences()`

  Updates all DITA references of the specified type. For example, you can use this API to update all `conrefs`, all `xrefs`, or all `conrefs` as well as `xrefs`.

Support for object styles

Object styles work similar to Paragraph and Character formats but work on objects. Users can save frequently used object properties as a style and can apply these object styles to various objects, such as images, anchored frames, and text frames for consistent size and appearance.

Object styles are controlled through the new `FO_GraphicsFmt` object.

Support for multiple views

To support the FrameMaker 11 feature of multiple views, FDK 11 provides a new session property `FP_ActiveView`, which can be set to one of the following:

- `WYSIWYG View`
- `Author View`
- `XML View`

Support for hotspots in objects

FDK 11 supports hotspots through the following properties:

- `FP_ViewHotspotIndicators`

  A document-level property that turns on hotspot indicators. A hotspot indicator is a square box at the centre of an object to indicate that the object is a hotspot.

- `FP_IsHotspot`

  An object-level property that specifies whether an object is a hotspot. If this property is turned off, the object is no longer considered a hotspot.

- `FP_HotspotCmdStr`

  The command string for the hotspot, similar to the hypertext command string. This is an object-level property.

- `FP_HotspotTitle`

  The tooltip for the hotspot. Applicable for the output formats that support it (for example HTML). This is an object-level property and is optional.

Interactive multimedia links for 3D objects

FDK 11 provides the following properties to manage links to embedded U3D (Universal 3D), FLV files, and SWF files in the PDF output.

- `FP_InsetGfxPlayWindowInPdf`

  If this property is set, on publishing a document to PDF, inset objects that have facets of type FLV, U3D, or SWF will be activated in a new Window in a PDF.

- `FP_InsetGfxActiveInPdf`

  If this property is set, on publishing a document to PDF, inset objects that have facets of type FLV, U3D, or SWF will be activated as soon as the page containing the graphic object is visible.

- `FP_InsetGfxName`

  This property assigns a name to inset objects that have facets of type FLV, U3D, or SWF.

- `FP_InsetJavaScriptAttached`

  Indicates whether Javascript is attached with the graphic object that has a U3D facet.

- `FP_InsetJavaScriptFile`

  Use this property to attach a Javascript file to a graphic object that has a U3D facet. If the value of the file path sepecified is null, the Javascript file attached to the inset is removed.

- `FP_InsetSaveFacetToFile`

  Saves the given facet of an inset to a given file.

- `FP_InsetU3dViewList`

  Returns the list of views defined in the U3D facet of an inset object.

- `FP_InsetU3dAnimationList`

  Returns the list of animations defined in the U3D facet of an inset object.

- `FP_InsetU3dPartList`

  Returns the list of parts defined in the U3D facet of an inset object.

- `FP_InsetMonikerFilePath`

  Returns the file path of the moniker of an inset object that has an OLE2 facet.

Support for referring to content using line numbers

FDK 11 supports the setting of line numbers. The line numbers are set at a document level and appear before the line. The following properties are provided for managing line numbers:

- `FP_LineNumRestart`

  If set, restarts line number display on each page.

- `FP_LineNumShow`

  If set, enables the line number display.

- `FP_LineNumDistance`

  Sets the line number display width, that is the space in which the line numbers are displayed.

Support for guided structured authoring with banner text

Banner text in a FrameMaker file instructs you about what to enter in an element. You can control the instructional text you want to display for each of the elements.The following properties are used to handle banner text:

- `FP_BannerText`

  Property of element definition object, `FO_ElementDef`. Users can set, change, or query the banner text associated with an element definition object.

- `FP_BannerTextDisplay`

  Boolean property of document specifying whether banner text should be displayed in a document.

New APIs, notifications, error codes, objects, and properties

The following new API, notifications, error codes, objects, and properties are added in this release:

New APIs

- `F_ApiClientPath()`
- `F_ApiDefineAndAddCommandEx()`
- `F_ApiDefineCommandEx()`

- F_ApiDeleteAllKeyDefinitions()
- F_ApiDeleteCondTag()
- F_ApiGetAllKeyDefinitions()
- F_ApiGetAllKeys()
- F_ApiGetKeyCatalog()
- F_ApiGetKeyDefinition()
- F_ApiGetNewXMLDefaultParams()
- F_ApiNewKeyCatalog()
- F_ApiNewKeyDefinition()
- F_ApiNewXML()
- F_ApiProgressBarEx()
- F_ApiSetAttributeSimple()
- F_ApiUpdateDITAReference()
- F_ApiUpdateDITAReferences()
- F_ApiUpdateKeyDefinition()

New notifications

- FA_Note_IsCommandEnabled
- FA_Note_LoadKeyCatalog
- FA_Note_PostSwitchView
- FA_Note_PreSwitchView
- FA_Note_ReLoadKeyCatalog
- FA_Note_UpdateDITAReference
- FA_Note_UpdateDITAReferences

New error codes

- FE_BadKey
- FE_BadKeyField
- FE_KeyCatalogIsStale
- FE_KeyCatalogNotLoaded

- FE_KeyDefinitionDoesNotExist
- FE_NonDITADocument
- FE_UpdateDITAReferenceFailed
- FE_UpdateDITAReferenceFailedCannotConvertToFMObject
- FE_UpdateDITAReferenceFailedCannotFindReferencedFile
- FE_UpdateDITAReferenceFailedCannotOpenReferencedFile
- FE_UpdateDITAReferenceFailedCannotResolveReference
- FE_UpdateDITAReferenceFailedInvalidElementType

New objects

- FO_GraphicsFmt
- FO_KeyCatalog

New properties

- FP_ActiveView
- FP_DefaultKeyCatalog
- FP_FirstDITAConrefElementInDoc
- FP_FirstDITALinkElementInDoc
- FP_FirstDITATopicrefElementInDoc
- FP_FirstDITATopicsetrefElementInDoc
- FP_FirstDITAXrefElementInDoc
- FP_FirstGraphicsFmtInDoc
- FP_FirstKeyCatalogInSession
- FP_FMConsoleString
- FP_Focus
- FP_GroupDialog
- FP_HotspotCmdStr
- FP_HotspotTitle
- FP_InsetMonikerFilePath
- FP_InsetMonikerPath

- FP_InsetU3dAnimationList
- FP_InsetU3dPartList
- FP_InsetU3dViewList
- FP_IsHotspot
- FP_IsTempOpenSave
- FP_KeyCatalog
- FP_KeyCatalogType
- FP_KeyCatalogWorkflow
- FP_LineNumDistance
- FP_LineNumRestart
- FP_LineNumShow
- FP_MaxSize
- FP_MinSize
- FP_NextDITAConrefElementInDoc
- FP_NextDITALinkElementInDoc
- FP_NextDITATopicrefElementInDoc
- FP_NextDITATopicsetrefElementInDoc
- FP_NextDITAXrefElementInDoc
- FP_NextGraphicsFmtInDoc
- FP_SpecifiedKeyCatalog
- FP_UseAFrameIsCropped
- FP_UseAFrameIsFloating
- FP_UseAlignment
- FP_UseAnchorType
- FP_UseAngle
- FP_UseBaselineOffset
- FP_UseBorderWidth
- FP_UseColGapWidth
- FP_UseColumnsAreBalanced
- FP_UseDTheta

- FP_UseFill
- FP_UseFlowIsAutoConnect
- FP_UseFlowIsPostScript
- FP_UseHeight
- FP_UseInsetDpi
- FP_UseLocX
- FP_UseLocY
- FP_UseMathSize
- FP_UseNumColumns
- FP_UseOverprint
- FP_UsePen
- FP_UseRadius
- FP_UseRunaround
- FP_UseRunaroundGap
- FP_UseSideHeadGap
- FP_UseSideHeadPlacement
- FP_UseSideHeadWidth
- FP_UseSideOffset
- FP_UseTextLineType
- FP_UseTheta
- FP_UseTintPercent
- FP_UseWidth
- FP_ViewHotspotIndicators
- FP_XSLTTransformationScenarioFile

CMS Connector Framework

FrameMaker extends the support to any standard content management system (CMS) by providing a set of full-featured APIs.
With CMS APIs, you can implement additional authoring and content management capabilities beyond the ones provided by the CMS.
In addition, you can create your own user interface or extend the existing user interface features.

New CMS API, structures, enums, error codes, enums, and properties added in this release:

- "CMS API Data Structures and Enum Constants" on page 993
- "Error Codes" on page 1000
- "CMS API Functions" on page 1001
- "APIs to automate the CMS connectors functionality" on page 1035

2

Function Summary

This chapter lists the FDK functions by topic. If you know what you want to do, but don't know which function to use, look it up in this chapter. If you know the name of a function and want a complete description of it, look it up in Chapter 2, "FDK Function Reference."

The functions are listed under all the topics to which they apply, so some functions appear more than once.

Client-defined callback functions

To	Use this function
Respond to the user choosing a menu item created by your client	`F_ApiCommand()`
Respond to the FrameMaker product's initialization message	`F_ApiInitialize()`
Respond to the user clicking a hypertext marker or inset	`F_ApiMessage()`
Respond to the user or another client executing operations such as Open or Save	`F_ApiNotify()`
Turn a notification on or off	`F_ApiNotification()`
Respond to the user interacting with a client-defined dialog box	`F_ApiDialogEvent()`
Set a callback function that is called for setting the username/password information before performing the NetLib related authentication.	`F_ApiNetLibSetAuthFunction()`
Set a return value for a client-defined callback function	`F_ApiReturnValue()`

Books

To	Use this function
Close a book	`F_ApiClose()`
Compare two books	`F_ApiCompare()`
Create a book	`F_ApiNewNamedObject()`
Generate/update files for a book	`F_ApiSimpleGenerate()` `F_ApiUpdateBook()`
Import element definitions to a book	`F_ApiSimpleImportElementDefs()`
Import formats from a document to a book	`F_ApiSimpleImportFormats()`
Insert a book component of a specified type at a specified position in a structured FrameMaker book.	`F_ApiNewBookComponentOfTypeInHierarchy()`
Move a book component within the book.	`F_ApiMoveComponent()`
Open a book (the easy way)	`F_ApiSimpleOpen()`

To	Use this function
Open a book (with a script)	`F_ApiOpen()`
Print a book	`F_ApiSilentPrintDoc()`
Save a book (the easy way)	`F_ApiSimpleSave()`
Save a book (with a script)	`F_ApiSave()`
Write entries to the book error log	`F_ApiCallClient()`

Characters

To	Use this function
Convert a character to lowercase	`F_CharToLower()`
Convert a character to uppercase	`F_CharToUpper()`
Convert a UTF-8 character to lowercase	`F_CharToLowerUTF8()`
Determine whether a character is alphabetic	`F_CharIsAlphabetic()`
Determine whether a character is alphanumeric	`F_CharIsAlphaNumeric()`
Determine whether a character is a FrameMaker product control character	`F_CharIsControl()`
Determine whether a character is alphabetic	`F_CharIsEol()`
Determine whether a character is hexadecimal	`F_CharIsHexadecimal()`
Determine whether a character is lowercase	`F_CharIsLower()`
Determine whether a character is numeric	`F_CharIsNumeric()`
Determine whether a character is uppercase	`F_CharIsUpper()`
Determine whether a specified UTF-8 character is an alphanumeric character	`F_CharIsAlphaNumericUTF8()`
Determine whether a specified character is a FrameMaker control character.	`F_CharIsControlUTF8()`
Determine whether a specified character is a FrameMaker end-of-line (EOL) character.	`F_CharIsEolUTF8()`
Determine whether a specified UTF-8 character is a hexadecimal digit	`F_CharIsHexadecimalUTF8()`
Determine whether a specified UTF-8 character is a lowercase character	`F_CharIsLowerUTF8()`
Determine whether a specified UTF-8 character is a numeric character in a decimal system.	`F_CharIsNumericUTF8()`
Determine whether a specified UTF-8 character is an uppercase character	`F_CharIsUpperUTF8()`

Clipboard

To	Use this function
Copy selection to the Clipboard	`F_ApiCopy()`
Clear selection	`F_ApiClear()`
Cut selection to the Clipboard	`F_ApiCut()`
Paste contents of the Clipboard	`F_ApiPaste()`
Pop the entry at the top of the Clipboard stack to the Clipboard	`F_ApiPopClipboard()`
Push the Clipboard contents onto the Clipboard stack	`F_ApiPushClipboard()`

Commands and menus

To	Use this function
Add a command to a menu	`F_ApiAddCommandToMenu()`
Add a menu to a menu or menu bar	`F_ApiAddMenuToMenu()`
Add a menu item separator to a menu	`F_ApiAddCommandToMenu()`
Allow user to choose a command by typing in the document window Tag area	`F_ApiQuickSelect()`
Arrange the menu items on a menu	`F_ApiSetId()`
Arrange the menus on a menu or menu bar	`F_ApiSetId()`
Create a command	`F_ApiDefineCommand()`
Create a command and specify the views in which this command is valid	`F_ApiDefineCommandEx()`
Create a command and add it to a menu	`F_ApiDefineAndAddCommand()`
Create a menu	`F_ApiDefineMenu()`
Create a menu and add it to a menu or menu bar	`F_ApiDefineAndAddMenu()`
Determine if a menu or command exists	`F_ApiGetNamedObject()`
Determine if a menu or menu item is on a menu or menu bar	`F_ApiMenuItemInMenu()`
Get the ID of a menu, command or menu item separator with a specified name	`F_ApiGetNamedObject()`

To	Use this function
Respond to the user executing a command created by your client	`F_ApiCommand()`
Load a menu customization file	`F_ApiLoadMenuCustomizationFile()`
Remove amenu item, menu item separator, menu, or command	`F_ApiDelete()`

Data structures: copying

To	Use this function
Copy an F_AttributesT structure	F_ApiCopyAttributes()
Copy an F_AttributeDefsT structure	F_ApiCopyAttributeDefs()
Copy an F_ElementCatalogEntriesT structure	F_ApiCopyElementCatalog Entries()
Copy an F_AttributesT structure	F_ApiCopyAttributes()
Copy an F_IntsT structure	F_ApiCopyInts()
Copy an F_MetricsT structure	F_ApiCopyMetrics()
Copy an F_PointsT structure	F_ApiCopyPoints()
Copy an F_PropValT structure	F_ApiCopyPropVal()
Copy an F_PropValsT structure	F_ApiCopyPropVals()
Copy a StringT variable	F_ApiCopyString()
Copy an F_StringsT structure	F_ApiCopyStrings()
Copy an F_TabT structure	F_ApiCopyTab()
Copy an F_TabsT structure	F_ApiCopyTabs()
Copy an F_TextItemT structure	F_ApiCopyTextItem()
Copy an F_TextItemsT structure	F_ApiCopyTextItems()
Copy an F_UBytesT structure	F_ApiCopyUBytes()
Copy an F_UIntsT structure	F_ApiCopyUInts()
Copy an F_ValT structure	F_ApiCopyVal()

Debugging

To	Use this function
Print the current API error status	F_ApiPrintFAErrno()
Print status flags returned by a call to F_ApiImport()	F_ApiPrintImportStatus()
Print status flags returned by a call to F_ApiOpen()	F_ApiPrintOpenStatus()

To	Use this function
Print the status flags returned by a call to `F_ApiSave()`	`F_ApiPrintSaveStatus()`
Print the status flags returned by a call to `F_ApiUpdateBook()`	`F_ApiPrintUpdateBookStatus()`
Print a property-value pair	`F_ApiPrintPropVal()`
Print the property-value pairs in a property list	`F_ApiPrintPropVals()`
Print the text in a single text item	`F_ApiPrintTextItem()`
Print the text in a set of text items (`F_TextItemsT` structure)	`F_ApiPrintTextItems()`

DITA references: updating

To	Use this function
Update the DITA object represented by the specified element.	`F_ApiUpdateDITAReference()`
Update all DITA references of the specified type. For example, you can use this API to update all `conrefs`, all `xrefs`, or all `conrefs` as well as `xrefs`.	`F_ApiUpdateDITAReferences()`

Dialog boxes: predefined

To	Use this function
Display an OK/Cancel or Continue dialog box with a specified message	`F_ApiAlert()`
Display a dialog box that allows the user to select a file or directory	`F_ApiChooseFile()`
Display a dialog box that prompts the user for an integer value	`F_ApiPromptInt()`
Display a dialog box that prompts the user for a metric value	`F_ApiPromptMetric()`
Display a dialog box that prompts the user for a string	`F_ApiPromptString()`
Display dialog boxes similar to FrameMaker's Open and Save dialog boxes	`F_ApiChooseFileExEx ()`
Display a scroll list that allows the user to choose from a list of items you provide	`F_ApiScrollBox()`

Dialog boxes: client-defined

To	Use this function
Close a dialog box	`F_ApiClose()`
Display a dialog resource as a modal dialog box	`F_ApiModalDialog()`
Display a dialog resource as a modeless dialog box	`F_ApiModelessDialog()`

To	Use this function
Get the ID of an item in a dialog box from its item number	`F_ApiDialogItemId()`
Get the string entered in a text field	`F_ApiGetString()`
Get the state of a button, checkbox, radio button, text box, pop-up menu, or scroll list	`F_ApiGetInt()`
Keep a client-defined modal dialog box on the screen after the user has clicked an item in it	`F_ApiReturnValue()`
Open a dialog resource	`F_ApiOpenResource()`
Respond to the user interacting with a client-defined dialog box	`F_ApiDialogEvent()`
Set the string entered in a text field	`F_ApiSetString()`
Set the list of strings that appears in a scroll list or pop-up menu	`F_ApiSetStrings()`
Set the state of a button, checkbox, radio button, text box, pop-up menu, or scroll list	`F_ApiSetInt()`

Documents: comparing

To	Use this function
Compare two documents	`F_ApiCompare()`

Documents: file operations

To	Use this function
Check status bit in status scripts returned by `F_ApiOpen()` or `F_ApiSave()`	`F_ApiCheckStatus()`
Close (quit) a document	`F_ApiClose()`
Create a new custom document	`F_ApiCustomDoc()`
Create a new document from a template	`F_ApiSimpleNewDoc()`
Create a new document using a script	`F_ApiOpen()`
Get a default property list for use with `F_ApiOpen()`	`F_ApiGetOpenDefaultParams()`
Get a default property list for use with `F_ApiSave()`	`F_ApiGetSaveDefaultParams()`
Get a default property list for use with `F_ApiImport()`	`F_ApiGetImportDefaultParams()`
Import a text or graphics file	`F_ApiImport()`
Open a document (the easy way)	`F_ApiSimpleOpen()`
Open a document (with a script)	`F_ApiOpen()`
Print a document	`F_ApiSilentPrintDoc()`
Save a document (the easy way)	`F_ApiSimpleSave()`
Save a document (with a script)	`F_ApiSave()`
Apply an attribute expression to a document to perform attribute-based-filtering.	`F_ApiApplyAttributeExpression()`
Apply an attribute-based expression to the document and the filtered text is displayed in the specified color.	`F_ApiPreviewAttributeExpression()`
Apply an attribute-based expression to the document where the filtered text is converted to conditional text.	`F_ApiApplyAttributeExpressionAsCondition()`

Documents: formatting

To	Use this function
Accept all the tracked changes in the specified document	`F_ApiTrackChangesAcceptAll()`
Apply a page's layout to another page	`F_ApiApplyPageLayout()`
Clear the change bars for a document	`F_ApiClearAllChangeBars()`
Redisplay a document after `FP_Displaying` has been turned off	`F_ApiRedisplay()`
Reformat a document after `FP_Reformatting` has been turned off	`F_ApiReformat()`
Rehyphenate words in a document	`F_ApiRehyphenate()`
Reject all the tracked changes in the specified document	`F_ApiTrackChangesRejectAll()`
Reset equation settings for a document	`F_ApiResetEqnSettings()`
Reset reference frames	`F_ApiResetReferenceFrames()`
Restart paragraph numbering for a document	`F_ApiRestartPgfNumbering()`

Documents: updating

To	Use this function
Import formats from another document	`F_ApiSimpleImportFormats()`
Import element definitions to a document	`F_ApiSimpleImportElementDefs()`
Updates a specified cross-reference in the document.	`F_ApiUpdateXRef()`
Update the cross-references in a document	`F_ApiUpdateXRefs()`
Update a text inset	`F_ApiUpdateTextInset()`
Update the variables in a document	`F_ApiUpdateVariables()`

F-codes

To	Use this function
Execute a set of f-codes	`F_ApiFcodes()`

Files, directories, and filepaths

To	Use this function
Copy a filepath (platform-independent representation of a pathname)	`F_FilePathCopy()`
Convert a filepath to a platform-specific or platform-independent pathname	`F_FilePathToPathName()`
Convert a platform-specific or platform-independent pathname to a filepath	`F_PathNameToFilePath()` `F_NewMacFilePath()`
Close a directory handle opened with `F_FilePathOpenDir()`	`F_FilePathCloseDir()`
Create a directory	`F_MakeDir()`
Delete a file specified by a filepath	`F_DeleteFile()`
Determine which platform naming conventions a pathname uses	`F_PathNameType()`
Free the memory used by a filepath	`F_FilePathFree()`
Get the basename (base directory name or filename) specified by a filepath	`F_FilePathBaseName()`
Get the filepath associated with a specified channel	`F_GetFilePath()`
Get the next file in a directory specified by a handle opened with `F_FilePathOpenDir()`	`F_FilePathGetNext()`
Get the parent directory of a specified file or directory	`F_FilePathParent()`
Get the permissions and other information for a specified file or directory	`F_FilePathProperty()` `F_GetMacFilePathInfo()`
Open and return a handle that can be used to obtain the file and subdirectory entries in a specified directory	`F_FilePathOpenDir()`
Rename a file	`F_RenameFile()`

To	Use this function
Reset a directory's handle so that the next call to `F_FilePathGetNext()` returns the first file or directory entry in the directory	`F_ResetDirHandle()`

Fonts

To	Use this function
Get the angles, variations, and weights available for a specified font or combined font	`F_ApiFamilyFonts()` `F_ApiCombinedFamilyFonts()`
Get the character encoding for a specified font or font family	`F_ApiGetEncodingForFamily()` `F_ApiGetEncodingForFont()`

Get and set character encoding data for a session	`F_ApiGetSupportedEncodings()`
	`F_ApiIsEncodingSupported()`
	`F_FontEncId()`
	`F_FontEncName()`

Graphic insets

To	Use this function
Create a graphic inset	`F_ApiImport()` `F_ApiNewGraphicObject()`
Delete a facet	`F_ApiDeletePropByName()`
Query an integer (`IntT`) facet	`F_ApiGetIntByName()`
Query a metric (`MetricT`) facet	`F_ApiGetMetricByName()`
Query an unsigned bytes (`F_UBytesT`) facet	`F_ApiGetUBytesByName()`
Respond to the user clicking an inset	`F_ApiMessage()`
Set an integer (`IntT`) facet	`F_ApiSetIntByName()`
Set a metric (`MetricT`) facet	`F_ApiSetMetricByName()`
Set an unsigned bytes (`F_UBytesT`) facet	`F_ApiSetUBytesByName()`

Hash tables

To	Use this function
Add an entry to a hash table	`F_HashSet()`
Call a specified function with the key and datum of each entry in a hash table	`F_HashEnumerate()`
Create a hash table	`F_HashCreate()`
Delete a hash table and all its entries	`F_HashDestroy()`
Get an entry from a hash table	`F_HashGet()`
Remove an entry from a hash table	`F_HashRemove()`
Create a hash table	`F_HashReportOnData()`

Hypertext

To	Use this function
Execute a hypertext command	`F_ApiHypertextCommand()`

I/O

To	Use this function
Append the contents of one channel to another	`F_ChannelAppend()`
Close a channel	`F_ChannelClose()`
Close a temporary channel	`F_ChannelCloseTmp()`
Create a temporary channel	`F_ChannelMakeTmp()`
Determine whether the end of a channel has been read	`F_ChannelEof()`
Flush buffered output to a channel	`F_ChannelFlush()`
Get the size of a channel	`F_ChannelSize()`
Get the byte after the current position in a channel	`F_ChannelPeek()`
Get the offset of the current byte relative to the beginning of an input channel	`F_ChannelTell()`
Open a channel	`F_ChannelOpen()`

To	Use this function

Read from a channel	`F_ChannelRead()`
	`F_ReadBytes()`
	`F_ReadLongs()`
	`F_ReadShorts()`

Read formatted input	`F_Scanf()` `F_Sscanf()`
Set the position for the next input operation in a channel	`F_ChannelSeek()`
Set the byte order of a channel so that subsequent I/O calls will swap bytes if the platform is big endian	`F_SetByteOrder()`
Set the byte order of a channel so that subsequent I/O calls will swap bytes if the platform is little endian	`F_ResetByteOrder()`

Write to a channel	`F_ChannelWrite()`
	`F_WriteBytes()`
	`F_WriteLongs()`
	`F_WriteShorts()`

Write formatted output	`F_Printf()` `F_Sprintf()`

Insets

See “Graphic insets” on page 41 and “Text insets” on page 64.

Key Catalogs

To	Use this function
Create a new key catalog	`F_ApiNewKeyCatalog()`
Find a key catalog with the specified tag.	`F_ApiGetKeyCatalog()`
Add a new key definition to the specified key catalog.	`F_ApiNewKeyDefinition()`
Update the specified key definition field for the specified key in the specified key catalog.	`F_ApiUpdateKeyDefinition()`
Get the specified key definition field for the specified key from the specified key catalog.	`F_ApiGetKeyDefinition()`
Get all the key definitions from the specified key catalog.	`F_ApiGetAllKeyDefinitions()`
Delete all the key definitions in the specified key catalog.	`F_ApiDeleteAllKeyDefinitions()`
Get all the key tags from the specified key catalog.	`F_ApiGetAllKeys()`

Memory: allocating and deallocating structures

To	Use this function
Allocate memory for a property list	`F_ApiAllocatePropVals()`
Allocate memory for text items (an `F_TextItemsT` structure)	`F_ApiAllocateTextItems()`
Exit and free all resources used by client	`F_ApiBailOut()`
Deallocate memory for a property (an `F_PropValT` structure)	`F_ApiDeallocateProp()`
Deallocate memory for a property list (an `F_PropValsT` structure)	`F_ApiDeallocatePropVals()`
Deallocate memory for text items (an `F_TextItemsT` structure)	`F_ApiDeallocateTextItems()`
Deallocate memory for an `F_FontsT` strcture	`F_ApiDeallocateFonts()`

To	Use this function
Deallocate memory for an `F_IntsT` structure	`F_ApiDeallocateInts()`
Deallocate memory for an `F_MetricsT` structure	`F_ApiDeallocateMetrics()`
Deallocate memory for an `F_PointsT` structure	`F_ApiDeallocatePoints()`
Deallocate memory for a `StringT` variable	`F_ApiDeallocateString()`
Deallocate memory for an `F_StringsT` structure	`F_ApiDeallocateStrings()`
Deallocate memory for an `F_TabT` structure	`F_ApiDeallocateTab()`
Deallocate memory for an `F_TabsT` structure	`F_ApiDeallocateTabs()`
Deallocate memory for an `F_UbytesT` structure	`F_ApiDeallocateUBytes()`
Deallocate memory for an `F_UIntsT` structure	`F_ApiDeallocateUInts()`

Memory: manipulating with handles

To	Use this function
Allocate a block of memory to a handle	`F_AllocHandle()`
Allocate a new block of memory to a handle and copy the contents of a specified block of memory to it	`F_DuplicateHandle()`
Compare two blocks of memory specified by handles	`F_HandleEqual()`
Free a block of memory specified by a handle	`F_FreeHandle()`
Get the size of a handle's block of data	`F_GetHandleSize()`
Initialize a block of memory specified by a handle to `0`	`F_ClearHandle()`
Lock a handle and return the address of the data block of the handle	`F_LockHandle()`
Reallocate a block of memory	`F_ReallocHandle()`
Specify a direct straight exit function for the FDE to call when memory allocation fails	`F_SetDSExit()`
Unlock a handle	`F_UnlockHandle()`

Memory: manipulating with pointers

To	Use this function
Allocate a block of memory to a pointer	`F_Alloc()` `F_Calloc()`
Allocate a new block of memory to a pointer and copy the contents of a specified block of memory to it	`F_DuplicatePtr()`
Compare two blocks of memory specified by pointers	`F_PtrEqual()`
Copy the contents of a block of memory to another block of memory	`F_CopyPtr()`
Free a block of memory specified by a pointer	`F_Free()`
Initialize a block of memory specified by a pointer to `0`	`F_ClearPtr()`

To	Use this function
Reallocate a block of memory to a pointer	`F_Realloc()`
Specify a direct straight exit function for the FDE to call when memory allocation fails	`F_SetDSExit()`

Menus

See “Commands and menus” on page 31.

Metrics

To	Use this function
Compare two metric numbers	`F_MetricApproxEqual()`
Compute the square root of a metric number	`F_MetricSqrt()`
Compute the square of a metric number	`F_MetricSquare()`
Constrain a specified angle to a specified range of degrees	`F_MetricConstrainAngle()`
Construct a metric number from a fraction	`F_MetricMake()`
Convert a metric number to a real number	`F_MetricToFloat()`
Convert a real number to a metric number	`F_MetricFloat()`
Divide two metric numbers	`F_MetricDiv()`
Multiply a metric value by a fraction	`F_MetricFractMul()`
Multiply two metric numbers	`F_MetricMul()`
Normalize a specified angle between 0 and 360 degrees	`F_MetricNormalizeAngle()`

Maker Interchange Format (MIF)

To	Use this function
Indent and start a new MIF statement and automatically increase the indent level	`F_MifBegin()`
Write a comment string to the MIF write channel	`F_MifComment()`
Write a real number with *n* digits after the decimal point	`F_MifDecimal()`
Indent and finish a MIF statement	`F_MifEnd()`
Return the current indent level of the MIF write channel	`F_MifGetIndent()`
Indent the output channel according to the indent level	`F_MifIndent()`
Decrease the indent level	`F_MifIndentDec()`
Increase the indent level	`F_MifIndentInc()`
Write an integer to the MIF write channel	`F_MifInteger()`

To	Use this function
Write a new line to the MIF write channel	`F_MifNewLine()`
Set the indent level of the MIF write channel	`F_MifSetIndent()`
Set a channel to receive MIF output	`F_MifSetOutputChannel()`
Write a blank space to the MIF output channel	`F_MifSpace()`
Write a tab to the MIF channel	`F_MifTab()`
Write a simple text string to the MIF output channel	`F_MifText()`
Write a text string enclosed in single quotes (`'`) to the MIF channel	`F_MifTextString()`

Objects: creating and deleting

To	Use this function
Add a Boolean conditional expression to the document	`F_ApiAddNewBuildExpr()` `F_ApiDeleteBuildExpr()`
Apply the Boolean conditional expression to the document.	`F_ApiSetActiveBuildExpr()`
Pre-register a given server with the provided username and password, so that instead of prompting the user for authentication, the server uses the login information when required.	`F_ApiNetLibAuthenticateServer()`
Return the name of the active expression in the document	`F_ApiGetActiveBuildExpr()`
Return the Boolean conditional expression in the document with the given name	`F_ApiGetBuildExpr()`
Return an array of all Boolean conditional expression names in the document	`F_ApiGetBuildExprCatalog()`
Upload a folder and subfolders to the server.	`F_ApiNetLibUploadFolder()`
Create an anchored object, such as a marker or an anchored frame	`F_ApiNewAnchoredObject()`
Create a book component in a structured book	`F_ApiNewBookComponentInHierarchy()`
Create a column in a table	`F_ApiAddCols()`
Create a client text inset	`F_ApiNewAnchoredObject()`
Create a format rule, format rule clause, or format change list	`F_ApiNewSubObject()`
Create a formatted anchored object, such as a table, variable, or cross-reference	`F_ApiNewAnchoredFormattedObject()`
Create a graphic object, such as a text column or an oval	`F_ApiNewGraphicObject()`
Create a named object, such as a reference page	`F_ApiNewNamedObject()`
Create a row in a table	`F_ApiAddRows()`

To	Use this function
Create a series object, such as a paragraph, body page, or book component	`F_ApiNewSeriesObject()`
Create a structural element in a Structured document	`F_ApiNewElement()`
Create a table with a specified number of rows and columns	`F_ApiNewTable()`
Create a structural element in a Structured document or book	`F_ApiNewElementInHierarchy()`
Create a text or graphic inset	`F_ApiImport()`
Delete unused formats (character, paragraph, or table) from the document	`F_ApiDeleteUnusedFmts()`
Delete an object	`F_ApiDelete()`

Objects: getting IDs

To	Use this function
Get the ID of an object with a specified persistent identifier	`F_ApiGetUniqueObject()`
Get the ID of an object that has a specified name	`F_ApiGetNamedObject()`
Query an ID (`F_ObjHandleT`) property	`F_ApiGetId()`

Printing

To	Use this function
Print a document	`F_ApiSilentPrintDoc()`
Specify the number of copies of a document to print and other integer print properties	`F_ApiSetInt()`
Specify the printer to which to print a document and other string print properties	`F_ApiSetString()`

Properties

To	Use this function
Get an element definition's attribute definitions	`F_ApiGetAttributeDefs()`
Get an element's attributes	`F_ApiGetAttributes()`
Get a document's element catalog	`F_ApiGetElementCatalog()`
Query a property of any type	`F_ApiGetPropVal()`
Query an array of integers (`F_IntsT`) property	`F_ApiGetInts()`
Query an ID (`F_ObjHandleT`) property	`F_ApiGetId()`
Query an integer (`IntT`) property	`F_ApiGetInt()`
Query a metric (`MetricT`) property	`F_ApiGetMetric()`
Query a metrics (`F_MetricsT`) property	`F_ApiGetMetrics()`
Query an array of points (`F_PointsT`) property for a polygon or polyline	`F_ApiGetPoints()`

To	Use this function
Query a string (StringT) property	F_ApiGetString()
Query a strings (F_StringsT) property	F_ApiGetStrings()
Query a text location (F_TextLocT) property	F_ApiGetTextLoc()
Query a text range (F_TextRangeT) property	F_ApiGetTextRange()
Query the text properties (for example, character tag, font family, font weight) for a text range	F_ApiGetTextProps() F_ApiGetTextPropVal() F_ApiGetTextVal()
Retrieve the array of tabs (F_TabsT) for a paragraph or paragraph format	F_ApiGetTabs()
Retrieve the entire property list (F_PropValsT) for an object	F_ApiGetProps()
Set an element definition's attribute definitions	F_ApiSetAttributeDefs()
Set an element's attributes	F_ApiSetAttributes()
Set a property of any type	F_ApiSetPropVal()
Set an array of integers (F_IntsT) property	F_ApiSetInts()
Set an ID (F_ObjHandleT) property	F_ApiSetId()
Set an integer (IntT) property	F_ApiSetInt()
Set a metric (MetricT) property	F_ApiSetMetric()
Set a metrics (F_MetricsT) property	F_ApiSetMetrics()
Set a text location (F_TextLocT) property	F_ApiSetTextLoc()
Set the entire property list (F_PropValsT) for an object	F_ApiSetProps()
Set an array of points (F_PointsT) property	F_ApiSetPoints()
Set a string (StringT) property	F_ApiSetString()
Set a strings (F_StringsT) property	F_ApiSetStrings()
Set a tabs (F_TabsT) property	F_ApiSetTabs()
Set the text properties (for example, character tag, font family, font weight) for a text range	F_ApiSetTextProps() F_ApiSetTextPropVal() F_ApiSetTextVal()
Set a text range (F_TextRangeT) property	F_ApiSetTextRange()

Selection

To	Use this function
Determine where the insertion point or selected text is	`F_ApiGetTextRange()`
Get the element selection in a Structured document	`F_ApiGetElementRange()`
Set the text selection or insertion point by setting the property that specifies the text selection (`FP_TextSelection`)	`F_ApiSetTextRange()`
Select cells in a table	`F_ApiMakeTblSelection()`
Get the IDs of selected graphic objects	`F_ApiGetId()`
Set the element selection in a Structured document	`F_ApiSetElementRange()`
Scroll the document to display the currenttrext range	`F_ApiScrollToText()`

Sessions

To	Use this function
Quit a session	`F_ApiClose()`

Sleep

To	Use this function
Suspend FrameMaker product and client operation for specified number of seconds	`F_ApiSleep()`
Suspend FrameMaker product and client operation for specified number of microseconds	`F_ApiUSleep()`

String lists

To	Use this function
Append a string to a string list	`F_StrListAppend()`
Concatenate two string lists	`F_StrListCat()`

To	Use this function
Copy a string from a string list to a string of a specified size	`F_StrListCopy()`
Copy an entire string list	`F_StrListCopyList()`
Create a string list	`F_StrListNew()`
Free a string list	`F_StrListFree()`
Get the th string in a string list	`F_StrListGet()`
Get the first string in a string list	`F_StrListFirst()`
Get the last string in a string list	`F_StrListLast()`
Get the position of a specified string in a string list	`F_StrListIndex()`
Get the position of a specified string in a string list, ignoring case	`F_StrListIIndex()`
Get the length of a string list	`F_StrListLen()`
Insert a string in a string list	`F_StrListInsert()`
Remove a specified string from a string list	`F_StrListRemove()`
Copy a string to a specified position in a string list	`F_StrListSetString()`
Sort a string list using a specified comparison function	`F_StrListSort()`

Strings: allocating, copying, and deallocating

To	Use this function
Allocate a new string	`F_StrNew()`
Copy a string, allocating memory for the new string	`F_StrCopyString()`
Copy a string to memory that is already allocated	`F_StrCpy()`
Copy a specified number of characters of a string	`F_StrCpyN()`
Deallocate a string	`F_Free()`

Strings: comparing and parsing

To	Use this function
Compare two strings	`F_StrCmp()` `F_StrEqual()`
Compare two strings, ignoring case	`F_StrICmp()` `F_StrIEqual()`
Compare two strings up to a specified number of characters	`F_StrEqualN()` `F_StrCmpN()`
Compare two strings up to a specified number of characters, ignoring case	`F_StrICmpN()` `F_StrIEqualN()`
Determine whether a string is a prefix of another string	`F_StrPrefix()`
Determine whether a string is a prefix of another string, up to a specified number of characters	`F_StrPrefixN()`
Determine whether a string is a prefix of another string, ignoring case	`F_StrIPrefix()`
Determine whether a string is a substring of another string	`F_StrSubString()`
Determine whether a string is a suffix of another string	`F_StrSuffix()`
Parse a string into tokens	`F_StrTok()`
Return a pointer to the first occurrence in a string of any of a specified set of characters	`F_StrBrk()`

To	Use this function
Return a pointer to the first occurrence of a character in a string	`F_StrChr()`
Return a pointer to the first occurrence of a character in a string, starting from the end of the string	`F_StrRChr()`

Strings: concatenating

To	Use this function
Concatenate two strings	`F_StrCat()`
Concatenate two strings, limiting the result to a specified number of characters	`F_StrCatN()`
Concatenate a string and an integer, limiting the result to a specified number of characters	`F_StrCatIntN()`
Concatenate a string and a character, limiting the result to a specified number of characters	`F_StrCatCharN()`

Strings: miscellaneous

To	Use this function
Convert a string to an integer	`F_StrAlphaToInt()`
Convert a string to a real number	`F_StrAlphaToReal()`
Get the length of a string	`F_StrLen()`
Reverse a string	`F_StrReverse()`
Strip a specified set of characters from a string	`F_StrStrip()`
Strip leading spaces from a string	`F_StrStripLeadingSpaces()`
Strip trailing spaces from a string	`F_StrStripTrailingSpaces()`
Truncate a string at a specified position	`F_StrTrunc()`

Strings: encoded

To	Use this function

Perform string operations on double-byte text that similar to the operations you can perform on single-byte text	`F_StrCatDblCharNEnc()`
	`F_StrCatNEnc()`
	`F_StrChrEnc()`
	`F_StrCmpNEnc()`
	`F_StrConvertEnc()`
	`F_StrCpyNEnc()`
	`F_StrICmpEnc()`
	`F_StrICmpNEnc()`
	`F_StrIEqualEnc()`
	`F_StrIEqualNEnc()`
	`F_StrIPrefixEnc()`
	`F_StrISuffixEnc()`
	`F_StrLenEnc()`
	`F_StrRChrEnc()`
	`F_StrTruncEnc()`

Get encoding information for a font or font family	`F_ApiGetEncodingForFamily()` `F_ApiGetEncodingForFont()`

Get and set encoding data for a session	`F_ApiGetSupportedEncodings()`
	`F_ApiIsEncodingSupported()`
	`F_FontEncId()`
	`F_FontEncName()`

Initialize encoding data for a session	`F_FdeInitFontEncs()`

Structure: manipulating elements

To	Use this function
Add an element to a location in the structural hierarchy of a Structured document or book	`F_ApiNewElementInHierarchy()`
Copy attributes	`F_ApiCopyAttribute()` `F_ApiCopyAttributes()`

To	Use this function
Copy newly added structures—perform a deep copy and copy any arrays or strings referenced by the structure.	`F_ApiCopyStructureTypes`
Create a book component in a structured book	`F_ApiNewBookComponentIn Hierarchy()`
Create an element in a document	`F_ApiNewElement()`
Deallocate attributes	`F_ApiDeallocateAttributes()`
Deallocate memory referenced by the newly added structures—perform a deep deallocation and deallocates any arrays or strings referenced by the structure.	F_ApiDeallocate*StructureType*()
Delete an element, but leave its contents in the document	`F_ApiUnWrapElement()`
Demote an element	`F_ApiDemoteElement()`
Get an element's attributes	`F_ApiGetAttributes()`
Get current valid elements for the insertion point	`F_ApiGetElementCatalog()`
Get the element selection	`F_ApiGetElementRange()`
Insert an element around the selected text and elements	`F_ApiWrapElement()`
Insert a book component of a specified type at a specified position in a structured FrameMaker book.	`F_ApiNewBookComponentOfTypeInHierarchy()`
Merge selected elements into the first element	`F_ApiMergeIntoFirst()`
Merge selected elements into the last element	`F_ApiMergeIntoLast()`
Promote an element	`F_ApiPromoteElement()`
Set the element selection in a Structured document or book	`F_ApiSetElementRange()`
Set an element's attributes	`F_ApiSetAttributes()`
Split one element into two	`F_ApiSplitElement()`

Structure: manipulating element definitions and format rules

To	Use this function
Check the value of an element definition to determine that the element is text and not a real element	`F_ApiElementDefIsText()`
Copy attribute definitions	`F_ApiCopyAttributeDef()` `F_ApiCopyAttributeDefs()`
Create a format change list (`FO_FmtChangeList` object)	`F_ApiNewNamedObject()` `F_ApiNewSubObject()`
Create a format rule (`FO_FmtRule` object)	`F_ApiNewSubObject()`
Create a format rule clause (`FO_FmtRuleClause` object)	`F_ApiNewSubObject()`
Deallocate attribute definitions	`F_ApiDeallocateAttributeDefs()`
Delete a format change list from the format change list catalog	`F_ApiDelete()`
Delete a format rule or format rule clause	`F_ApiDelete()`
Get an element definition's attribute definitions	`F_ApiGetAttributeDefs()`
Get the ID of a format change list	`F_ApiGetNamedObject()`
Import element definitions to a document	`F_ApiSimpleImportElementDefs()`
Set an element definition's attribute definitions	`F_ApiSetAttributeDefs()`

Tables

To	Use this function
Add columns to a table	`F_ApiAddCols()`
Add rows to a table	`F_ApiAddRows()`
Create a new table	`F_ApiNewTable()`
Delete columns from a table	`F_ApiDeleteCols()`
Delete rows from a table	`F_ApiDeleteRows()` `F_ApiDelete()`
Delete a table	`F_ApiDelete()`

To	Use this function
Select cells in a table	`F_ApiMakeTblSelection()`
Straddle cells in a table	`F_ApiStraddleCells()`
Unstraddle cells in a table	`F_ApiUnStraddleCells()`

Text

To	Use this function
Clear text selection from a document	`F_ApiClear()`
Copy text selection from a document to the Clipboard	`F_ApiCopy()`
Cut text selection from a document to the Clipboard	`F_ApiCut()`
Delete text from a paragraph or graphic text line	`F_ApiDeleteText()`
Get the text (array of text items) that an object contains	`F_ApiGetText()`
Get the text (array of text items) that an text range contains	`F_ApiGetTextForRange()`
Get a text property (for example, character tag, font family, font weight) for a text location	`F_ApiGetTextPropVal()` `F_ApiGetTextVal()`
Get all the text properties for a text location	`F_ApiGetTextProps()`
Insert text in a paragraph, graphic text line, or client text inset	`F_ApiAddText()`
Paste Clipboard contents at insertion point in a document	`F_ApiPaste()`
Query a text range property	`F_ApiGetTextRange()`
Set a text property for a text range	`F_ApiSetTextPropVal()` `F_ApiSetTextVal()`
Set all the text properties for a text range	`F_ApiSetTextProps()`
Set a text range property	`F_ApiSetTextRange()`

Text: find and replace

To	Use this function
Execute the find and replace command from an API client	`F_ApiFind()`

Text insets

To	Use this function
Add text to a client text inset (`FO_TiApiClient` object)	`F_ApiAddText()`
Convert a locked text inset range to text	`F_ApiConvertToText()`
Create a Frame product text inset (`FO_TiFlow`, `FO_TiText`, or `FO_TiTextTable` object)	`F_ApiImport()`
Create a client text inset	`F_ApiNewAnchoredObject()`
Delete a text inset	`F_ApiDelete()`
Delete text from a client text inset (`FO_TiApiClient` object)	`F_ApiDeleteTextInsetContents()`
Get a default property list for use with `F_ApiImport()`	`F_ApiGetImportDefaultParams()`
Get text from a text inset	`F_ApiGetText()`
Update stale text insets	`F_ApiUpdateTextInset()`

Unicode

To	Use this function
Enable *Unicode Mode* or *Compatibility Mode*	`F_ApiEnableUnicode()`

Undo/Redo

To	Use this function
Clear both the undo and redo stacks in the document specified by docId.	`F_ApiUndoCancel()`
Mark the starting point of a series of API calls that are to be treated as a single undoable operation in the document specified by docId.	`F_ApiUndoStartCheckPoint()`

To	Use this function
Mark the ending point of a series of API calls that are to be treated as a single undoable operation for the docid specified in the call to F_ApiUndoStartCheckpoint	`F_ApiUndoEndCheckPoint()`

Utility

To	Use this function
Allow the user to cancel a client operation	`F_ApiUserCancel()`
Call another FDK client	`F_ApiCallClient()`
Determine if an ID represents a valid object in a specified document	`F_ApiObjectValid()`
Exit and free memory	`F_ApiBailOut()`
Get a default property list for use with `F_ApiImport()`	`F_ApiGetImportDefaultParams()`
Get a default property list for use with `F_ApiOpen()`	`F_ApiGetOpenDefaultParams()`
Get a default property list for use with `F_ApiSave()`	`F_ApiGetSaveDefaultParams()`
Get an object's type	`F_ApiGetObjectType()`
Get the index for a property-value pair (`PropValT`) in a property list	`F_ApiGetPropIndex()`
Get the directory containing the current client	`F_ApiClientDir()`
Get the registered name of the current client	`F_ApiClientName()`
Reformat a document	`F_ApiReformat()`
Request notification for specific events from the Frame product	`F_ApiNotification()`
Suspend Frame product and client operation for specified number of seconds	`F_ApiSleep()`
Set the default directory for the client	`F_ApiSetClientDir()`
Suspend Frame product and client operation for specified number of microseconds	`F_ApiUSleep()`

3

FDK Function Reference

This chapter lists the FDK functions alphabetically. If you know the name of a function and want a complete description of it, look it up in this chapter. If you know what you want to do, but don't know which function to use, see Chapter 1, "Function Summary."

Some examples in this chapter use FDE functions, such as `F_Alloc()` and `F_Sprintf()`. For more information on using FDE and FDE functions, see Part III, "Frame Development Environment (FDE)," in the *FDK Programmer's Guide*.

IMPORTANT: *The examples in this chapter assume that your client includes the* `fapi.h` *and* `fdetypes.h` *header files in addition to the header files listed in the function synopsis.*

F_Alloc()

`F_Alloc()` allocates a block of memory.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
PtrT F_Alloc(UIntT n,
   PUCharT flags);
```

Arguments

`n`	The number of bytes of memory to allocate
`flags`	Specifies whether to bail out (`DSE`) or return `NULL` (`NO_DSE`) if the requested memory isn't available

Returns

A pointer to the allocated block of memory, or `NULL` if the requested memory isn't available.

Example

The following code allocates memory to a pointer, clears it, and then frees it:

```
. . .
UCharT *ptr = NULL;

ptr = F_Alloc(65535, NO_DSE);
if(ptr == NULL)
{
   F_Printf(NULL, "Couldn't Allocate memory.\n");
   return;
}
else
   F_ClearPtr(ptr, 65535);
. . .
F_Free(ptr);
. . .
```

See also

"F_AllocHandle()" on page 69.

F_AllocHandle()

`F_AllocHandle()` allocates a new handle to a block of memory. Use `F_FreeHandle()` to free the memory when you are done with it.

After you have allocated a handle, call `F_LockHandle()` to get the address of the handle's block of memory.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
HandleT F_AllocHandle(UIntT n,
    PUCharT flags);
```

Arguments

`n`	The number of bytes of memory to allocate
`flags`	Specifies whether to bail out (`DSE`) or return `NULL` (`NO_DSE`) if the requested memory isn't available

Returns

A handle to the allocated block of memory, or `NULL` if the requested memory isn't available.

Example

The following code allocates a block of memory to a handle, clears the memory, and then stores some data to it. After storing data to the memory, it unlocks and frees the handle.

```
. . .
HandleT hndl = NULL;
UCharT *ptr;
UIntT i;

hndl = F_AllocHandle(66000, NO_DSE);
if(hndl == NULL)
{
    F_Printf(NULL, "Couldn't allocate handle.\n");
    return;
}
F_ClearHandle(hndl);
ptr = F_LockHandle(hndl);

/* Stuff the block of memory with a bunch of 9s. */
for(i= 0; i < 66000; i++)
    ptr[i] = '9';

F_FreeHandle(hndl);
. . .
```

See also

"F_Alloc()" on page 68, "F_LockHandle()" on page 577, and "F_UnlockHandle()" on page 714.

F_ApiAddCols()

`F_ApiAddCols()` adds columns to a table.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiAddCols(F_ObjHandleT docId,
    F_ObjHandleT tableId,
    IntT refColNum,
    IntT direction,
    IntT numNewCols);
```

Arguments

`docId`	The ID of the document containing the table.
`tableId`	The ID of the table.
`refColNum`	The column at which to start adding columns. The columns are numbered from left to right starting with column `0`.
`direction`	The direction from the reference column in which to add columns. To add columns to the left of the reference column, specify `FV_Left`. To add them to the right, specify `FV_Right`.
`numNewCols`	The number of columns to add.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiAddCols()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadParameter`	The function call specified an invalid parameter
`FE_BadObjId`	Invalid object ID
`FE_BadOperation`	The function call specified an illegal operation

Example

The following code adds a column to the right of the first column in the selected table:

```
. . .
F_ObjHandleT docId, tblId;

/* Get the document and table IDs. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tblId = F_ApiGetId(FV_SessionId, docId, FP_SelectedTbl);

/* Add the column. */
F_ApiAddCols(docId, tblId, 0, FV_Right, 1);
. . .
```

See also

“F_ApiAddRows()” on page 78.

F_ApiAddCommandToMenu()

`F_ApiAddCommandToMenu()` adds a FrameMaker product command or a client-defined command to a menu.

`F_ApiAddCommandToMenu()` adds the command at the bottom of the specified menu. To change a command's position on a menu, set its `FP_PrevMenuItemInMenu` and `FP_NextMenuItemInMenu` properties. For more information on arranging menus and menu items, see "Arranging menus and menu items" on page 388 of the *FDK Programmer's Guide*.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiAddCommandToMenu(F_ObjHandleT toMenuId,
    F_ObjHandleT commandId);
```

Arguments

`toMenuId`	The ID of the menu to which to add the command
`commandId`	The ID of the command

To add a command that you have created, set `commandId` to the ID returned by the `F_ApiDefineCommand()` or `F_ApiDefineCommandEx()` call that created the command.

To add a FrameMaker product command, you must get its ID. To get its ID, call `F_ApiGetNamedObject()` with the `objectName` parameter set to its name. For example, to get the ID of the Copy command, use the following code:

```
. . .
F_ObjHandleT copyCmdId;
copyCmdId = F_ApiGetNamedObject(FV_SessionId, FO_Command,
                        "Copy");
. . .
```

To add a command to a menu that you have created, set `toMenuId` to the ID returned by `F_ApiDefineMenu()` or `F_ApiDefineAndAddMenu()` when you created the menu.

To add a menu item to a FrameMaker product menu, you must get the menu's ID by calling `F_ApiGetNamedObject()` with the `objectName` parameter set to the menu's name.

The [Files] section of the `maker.ini` file specifies the location of the menu and command configuration files that list FrameMaker's menus and commands.

Users can create custom menu files that override the default menus and commands. For more information, see your FrameMaker product installation documentation.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiAddCommandToMenu()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_BadOperation`	Parameters specify an invalid operation
`FE_NotCommand`	`commandId` doesn't specify a command
`FE_NotMenu`	`toMenuId` doesn't specify a menu
`FE_BadParameter`	Parameter has an invalid value
`FE_SystemError`	System error

Example

The following code creates a command named MyCommand and adds it to the File menu. It also adds the FrameMaker product command `UpperCaseText` to the File menu. It then rearranges the commands on the menu so that the two newly added commands appear first:

```
. . .
#define MY_CMD 1
F_ObjHandleT cmdId, ucCmdId, menuId;

/* Create the command. */
cmdId = F_ApiDefineCommand(MY_CMD, "MyCommand",
              "My Command", "\\!MC");

/* Get the ID of the file menu and the UpperCaseText command. */
menuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "FileMenu");
ucCmdId = F_ApiGetNamedObject(FV_SessionId,
                    FO_Command, "UpperCaseText");

/* Add the two commands to the File menu. */
F_ApiAddCommandToMenu(menuId, cmdId);
F_ApiAddCommandToMenu(menuId, ucCmdId);

/* Set command object properties to rearrange menu items. */
F_ApiSetId(FV_SessionId, menuId, FP_FirstMenuItemInMenu, cmdId);
F_ApiSetId(menuId, ucCmdId, FP_PrevMenuItemInMenu, cmdId);
. . .
```

See also

"F_ApiDefineCommand()" on page 137, "F_ApiDefineAndAddCommand()" on page 126, and "F_ApiGetNamedObject()" on page 214.

F_ApiAddMenuToMenu()

`F_ApiAddMenuToMenu()` adds a FrameMaker product menu or a menu that you have created to another menu or menu bar.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiAddMenuToMenu(F_ObjHandleT toMenuId,
   F_ObjHandleT menuId);
```

Arguments

`toMenuId`	The ID of the menu to which the new menu is to be added
`menuId`	The ID of the new menu

To add a menu to a menu or menu bar that you have created, set `toMenuId` to the ID returned by the `F_ApiDefineMenu()` or `F_ApiDefineAndAddMenu()` call that created the menu or menu bar.

To add a menu to one of the FrameMaker product's menus or menu bars, you must get the menu or menu bar's ID. To get its ID, call `F_ApiGetNamedObject()` with the `objectName` parameter set to its name. For example, to get the ID of the Book main menu bar, use the following code:

```
. . .
F_ObjHandleT mainMenuId;
mainMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                        "!BookMainMenu");
. . .
```

The [Files] section of the maker.ini file specifies the location of the menu and command configuration files that list FrameMaker's menus and commands.

IMPORTANT: *Your menu appears only on the menu bar you specify. For example, if you only add a menu to the* `!MakerMainMenu` *menu bar, the menu will not appear if the user switches to quick menus. For your menu to appear after the user has switched to quick menus, you must also add it to* `!QuickMakerMainMenu`.

The following table lists the types of menus you can add a menu to and how the FrameMaker product implements the added menu.

Type of menu or menu bar you are adding a menu to	How the FrameMaker product implements the added menu	Where the FrameMaker product adds the menu
Menu bar	Pull-down menu	At the right of the menu bar
Pull-down menu	Pull-right menu	At the bottom of the pull-down menu

Type of menu or menu bar you are adding a menu to	How the FrameMaker product implements the added menu	Where the FrameMaker product adds the menu
Pop-up menu	Pull-right menu	At the bottom of the pop-up menu
Pull-right menu	Pull-right menu	At the bottom of the pull-right menu

To change a menu's position on a menu or menu bar after you add it, set its `FP_PrevMenuItemInMenu` and `FP_NextMenuItemInMenu` properties. For more information on arranging menus and menu items, see "Arranging menus and menu items" on page 388 of the *FDK Programmer's Guide*.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiAddMenuToMenu()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_NotMenu`	`toMenuId` and `menuId` don't specify menus
`FE_BadOperation`	Parameters specify an invalid operation
`FE_BadParameter`	Parameter has an invalid value
`FE_SystemError`	System error

Example

The following code creates a menu with one command and adds it to the FrameMaker main menu bar (for both complete and quick menus):

```
. . .
#define MY_CMD 1
F_ObjHandleT quickMenuBarId, menubarId, menuId, cmdId;

/* Create the menu, then create a command and add it. */
menuId = F_ApiDefineMenu("APIMenu", "API Menu");
cmdId = F_ApiDefineAndAddCommand(MY_CMD, menuId, "MyCmd",
                    "My Cmd", "");

/* Get ID of FrameMaker main menu (complete menus). */
menubarId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                    "!MakerMainMenu");

/* Get ID of FrameMaker main menu (quick menus). */
quickMenuBarId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                    "!QuickMakerMainMenu");

/* Add the menu to the menu bars. */
F_ApiAddMenuToMenu(menubarId, menuId);
F_ApiAddMenuToMenu(quickMenuBarId, menuId);
. . .
```

See also

"F_ApiDefineMenu()" on page 141, "F_ApiDefineAndAddMenu()" on page 134, and "F_ApiGetNamedObject()" on page 214.

F_ApiAddRows()

`F_ApiAddRows()` adds one or more rows to a table.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiAddRows(F_ObjHandleT docId,
    F_ObjHandleT refRowId,
    IntT direction,
    IntT numNewRows);
```

Arguments

`docId`	The ID of the document containing the table.
`refRowId`	The ID of the row at which to starting adding rows.
`direction`	The direction from the reference row in which to add rows. For a list of the constants you can specify, see the table below.
`numNewRows`	The number of rows to add.

The following table lists the constants you can specify for `direction`.

Direction constant	Meaning
`FV_Above`	Add rows above the row specified by `refRowId`. The added rows are the same type as the row specified by `refRowId`.
`FV_Below`	Add rows below the row specified by `refRowId`. The added rows are the same type as the row specified by `refRowId`.
`FV_Body`	Add body rows at the bottom of the existing body rows (`refRowId` is used to determine which table to add rows to).
`FV_Footing`	Add footing rows at the bottom of the existing footing rows (`refRowId` is ignored).
`FV_Heading`	Add heading rows at the bottom of the existing heading rows (`refRowId` is ignored).

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiAddRows()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid row ID
`FE_BadOperation`	Parameters specify an invalid operation
`FE_BadParameter`	Parameter has an invalid value

Example

The following code adds two rows after the first row of the selected table:

```
. . .
F_ObjHandleT docId, tblId, row1Id;

/* Get the document and table IDs. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tblId = F_ApiGetId(FV_SessionId, docId, FP_SelectedTbl);

/* Get the ID for row 1. */
row1Id = F_ApiGetId(docId, tblId, FP_FirstRowInTbl);

/* Add the rows. */
F_ApiAddRows(docId, row1Id, FV_Below, 2);
. . .
```

See also

"F_ApiAddCols()" on page 70.

F_ApiAddText()

`F_ApiAddText()` inserts text into a paragraph or a text line.

Synopsis

```
#include "fapi.h"
. . .
F_TextLocT F_ApiAddText(F_ObjHandleT docId,
    F_TextLocT *textLocp,
    StringT text);
```

Arguments

`docId`	The ID of the document
`textLocp`	The text location at which to add the text
`text`	The text to add

The text you specify for `text` must use the FrameMaker product character set. To add special characters, you must specify octal (\) or hexadecimal (\x) sequences. The following table lists some of these sequences.

Special character	Hexadecimal representation	Octal representation
>	`\x3e`	`\76`
" (straight double quotation mark)	`\x22`	`\42`
“ (left double quotation mark)	`\xd2`	`\322`
” (right double quotation mark)	`\xd3`	`\323`

For a complete list of the characters in the FrameMaker product character set and the corresponding hexadecimal codes, see your Frame product user's manual. If you are not using ANSI C, you must specify octal (\) sequences instead of hexadecimal codes. If you are using ANSI C, you can specify either octal or hexadecimal sequences.

Returns

The text location of the end of the text that was added. If it fails, the returned text location is invalid.

If `F_ApiAddText()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID.
`FE_BadObjId`	Invalid object ID.
`FE_NotTextObject`	The object that `textLocp` specifies is not a paragraph (`FO_Pgf`) or a text line (`FO_TextLine`).
`FE_OffsetNotFound`	The offset specified for the text location couldn't be found in the specified text object.
`FE_ReadOnly`	The specified document is read-only.
`FE_BadSelectionForOperation`	The location that `textLocp` specifies is invalid. For example, it is inside a variable or outside the highest level element in a structured document.

Example

To add the following text at the insertion point (or the end of the current text selection),

```
The new CoffeeTool
is now on sale.
```

use the following code:

```
. . .
F_TextLocT trm;
F_TextRangeT tr;
F_ObjHandleT docId;

/* Get current text selection. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId,docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Add text. Use octal 011 for Return (end of paragraph). */
trm = F_ApiAddText(docId, &tr.end, "The new CoffeeTool\011");

/* Add more text to the end of the text that was just added. */
F_ApiAddText(docId, &trm, "is now on sale.");
. . .
```

See also

"F_ApiGetText()" on page 249 and "F_ApiDeleteText()" on page 151.

F_ApiAlert()

`F_ApiAlert()` displays an alert box with a message. Depending on the constant you specify for `type`, it displays either OK and Cancel buttons, Yes and No buttons, or a Continue button.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiAlert(StringT message, IntT type);
```

Arguments

`message`	The message that appears in the dialog box. If the message is longer than 255 characters, it is truncated.
`type`	The dialog box type. See below for dialog types.

Specify one of the following constants for the `type` argument:

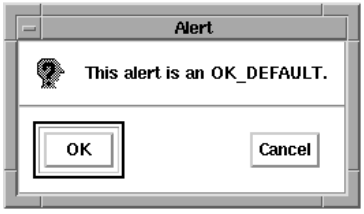
type constant	Type of dialog box displayed
`FF_ALERT_OK_DEFAULT`	Displays OK and Cancel buttons; OK is the default
`FF_ALERT_CANCEL_DEFAULT`	Displays OK and Cancel buttons; Cancel is the default
`FF_ALERT_CONTINUE_NOTE`	Displays Ok button
`FF_ALERT_CONTINUE_WARN`	Displays Ok button with a warning indication
`FF_ALERT_YES_DEFAULT`	Displays Yes and No buttons; Yes is the default
`FF_ALERT_NO_DEFAULT`	Displays Yes and No buttons; No is the default

Returns

`0` if the user clicked OK, Continue, or Yes; `-1` if the user clicked Cancel or No.

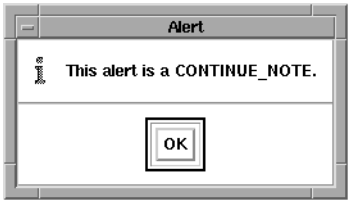
Example

Use the code shown in Figure 2-1 and Figure 2-2 to produce the following alert boxes.




```
F_ApiAlert("This alert is an OK_DEFAULT.", FF_ALERT_OK_DEFAULT);
```

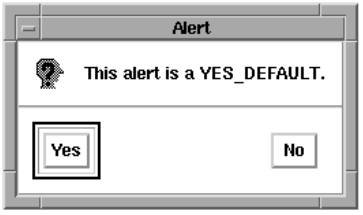
```
F_ApiAlert("This alert is a CANCEL_DEFAULT.", FF_ALERT_CANCEL_DEFAULT);
```



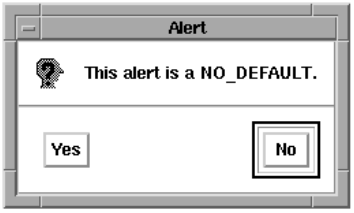

```
F_ApiAlert("This alert is a CONTINUE_NOTE.", FF_ALERT_CONTINUE_NOTE);
```

```
F_ApiAlert("This alert is a CONTINUE_WARN.", FF_ALERT_CONTINUE_WARN);
```

Figure 2-1 *Sample code for alert boxes*




```
F_ApiAlert("This alert is a YES_DEFAULT.", FF_ALERT_YES_DEFAULT);
```




```
F_ApiAlert("This alert is a NO_DEFAULT.", FF_ALERT_NO_DEFAULT);
```

Figure 2-2 *Sample code for alert boxes*

F_ApiAlive()

Checks whether the current asynchronous client has a connection with a FrameMaker process. Use it after registering the asynchronous client via `F_ApiWinConnectSession()`.

Synopsis

```
#include "f_stdio.h"
. . .
IntT F_ApiAlive (VoidT);
```

Returns

If there is a current connection to a FrameMaker process, this function returns a positive integer. Otherwise it returns 0.

See also

F_ApiWinConnectSession()

F_ApiAllocatePropVals()

`F_ApiAllocatePropVals()` allocates memory for a property list.

Synopsis

```
#include "fapi.h"
. . .
F_PropValsT F_ApiAllocatePropVals(IntT numProps);
```

Arguments

`numProps` The number of properties in the property list

Returns

A property list (an `F_PropValsT` data structure).

The returned `F_PropValsT` structure references memory that is allocated by the API. Use `F_ApiDeallocatePropVals()` to free this memory when you are done with it.

If `F_ApiAllocatePropVals()` fails, the API sets the `len` field of the returned structure to `0`.

Example

The following code allocates memory for an Open script (property list):

```
. . .
F_PropValsT openParams;

openParams = F_ApiAllocatePropVals(FS_NumOpenParams);

if(openParams.len == 0) return;

/* Use property list here. */

/* We're done with the property list, so we deallocate it. */
F_ApiDeallocatePropVals(&openParams);
. . .
```

See also

"F_ApiDeallocateStructureType()" on page 125.

F_ApiAllocateTextItems()

`F_ApiAllocateTextItems()` allocates memory for an `F_TextItemsT` structure and the array of text items that it references.

Synopsis

```
#include "fapi.h"
. . .
F_TextItemsT F_ApiAllocateTextItems(IntT numTextItems);
```

Arguments

`numTextItems`	The number of text items to allocate

Returns

An `F_TextItemsT` structure.

The returned `F_TextItemsT` structure references memory that is allocated by the API. Use `F_ApiDeallocateTextItems()` to free this memory when you are done with it.

If `F_ApiAllocateTextItems()` fails, the API sets the `len` field of the returned structure to `0`.

Example

The following code allocates ten text items:

```
. . .
F_TextItemsT tis;

tis = F_ApiAllocateTextItems(10);
. . .
```

See also

"F_ApiGetText()" on page 249.

F_ApiApplyAttributeExpression()

Applies an attribute expression to a document to perform attribute-based-filtering.

Synopsis

```
#include "fapi.h"
...
IntT F_ApiApplyAttributeExpression(F_ObjHandleT docId,
F_ObjHandleT attrExprId);
```

Arguments

`docId`	The ID of the document to which the attribute expression is to be applied.
`attrExprId`	The ID of the attribute expression to be applied to the document.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs. If `F_ApiApplyAttributeExpression()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno Value	Meaning
`FE_BadDocId`	Invalid document ID.
`FE_BadObjId`	Invalid expression ID.
`FE_InvalidAttrExpr`	The attribute expression being invalid cannot be applied to the document
`FE_WrongProduct`	The current FrameMaker product doesn't support the operation.
`FE_SystemError .`	Couldn't allocate memory

Example

The following code applies the expression named 'WebOutput' to the document.

```
. . .
F_ObjHandleT attrExprId = F_ApiGetNamedObject(docId,
                           FO_AttrCondExpr,"WebOutput");
F_ApiApplyAttributeExpression(docId, attrExprId);
...
```

F_ApiApplyAttributeExpressionAsCondition()

Applies an attribute-based expression to the document where the filtered text is converted to conditional text.

Synopsis

```
#include "fapi.h"
...
StatusT F_ApiApplyAttributeExpressionAsCondition(
    F_ObjHandleT docId,
    F_ObjHandleT attrExpId,
    F_ObjHandleT condId,
    BoolT removePreviouslyApplied);
```

Arguments

`docId`	The ID of the document on which attrExpId is to be applied
`attrExpId`	The ID of the attribute expression to be applied on the document
`condId`	The ID of the conditional text format that will be applied on the filtered text
`removePreviouslyApplied`	If true, then remove the conditional text settings at all locations in the document, whereever condId is applied.

Returns

If succeeds: `FE_Success`

If an error occurs: An error code

If API fails, the API assigns one of the following values to `FA_errno`:

FA_errno value	Meaning
`FE_Success`	Operation is successful
`FE_BadDocId`	Invalid document ID
`FE_InvalidAttrExpr`	The expression string set in the attrExpId's object is invalid
`FE_WrongProduct`	Current product interface isn't Structured FrameMaker
`FE_BadObjId`	Invalid object ID

F_ApiApplyPageLayout()

`F_ApiApplyPageLayout()` applies a page's layout to another page.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiApplyPageLayout(F_ObjHandleT docId,
   F_ObjHandleT destPage,
   F_ObjHandleT srcPage);
```

Arguments

`docId`	The document containing the pages
`destPage`	The page with the layout to apply
`srcPage`	The page to which to apply the layout

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiApplyPageLayout()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_BadOperation`	Parameters specify an invalid operation
`FE_BadParameter`	Parameter has an invalid value
`FE_SystemError`	System error

Example

The following code applies the layout of the Right master page to the current page:

```
. . .
F_ObjHandleT docId, curPageId, rtPageId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
curPageId = F_ApiGetId(FV_SessionId, docId, FP_CurrentPage);
rtPageId = F_ApiGetId(FV_SessionId, docId, FP_RightMasterPage);
F_ApiApplyPageLayout(docId, curPageId, rtPageId);
. . .
```

See also

"F_ApiSimpleImportFormats()" on page 462.

F_ApiBailOut()

`F_ApiBailOut()` allows your client to bail out. When your client calls `F_ApiBailOut()`, the FrameMaker product waits until it returns control from the current callback, and then exits it, saving system resources.

After your client has bailed out, the FrameMaker product still processes events that affect it. Menus your client created are still valid. If your client has requested notification for particular events, the FrameMaker product continues to monitor those events. It also monitors `apiclient` hypertext commands that specify your client's name. If these events occur, the FrameMaker product restarts your client by calling its `F_ApiInitialize()` function with `initialization` set to `FA_Init_Subsequent`.

IMPORTANT: *If your client bails out, it loses all its global variable settings.*

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiBailOut();
```

Arguments

None.

Returns

`VoidT`.

If `F_ApiBailOut()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

The following code bails out if the FrameMaker product currently running is not FrameMaker:

```
. . .
StringT product;

product = F_ApiGetString(0, FV_SessionId, FP_ProductName);
if (!F_StrEqual(product, "FrameMaker"))
{
    F_ApiAlert ("FDK client requires FrameMaker to run.",
                  FF_ALERT_CONTINUE_WARN);
    F_ApiBailOut();

    /* Bail out is not effective until return. */
    return;
}
. . .
```

F_ApiCallClient()

`F_ApiCallClient()` allows a client to call another client. It is useful for calling Structured FrameMaker clients, such as the structure generator and the element catalog manager.

`F_ApiCallClient()` calls the `F_ApiNotify()` function of the target client, with `notification` set to `FA_Note_ClientCall`, `docId` set to `0`, and `sparm` set to the string specified by `arg`. To receive the notification sent by `F_ApiCallClient()`, the target client must be registered and must have requested the `FA_Note_ClientCall` notification, as shown in the following code:

```
. . .
F_ApiNotification(FA_Note_ClientCall, True);
. . .
```

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiCallClient(StringT clname,
   StringT arg);
```

Arguments

`clname`	The registered name of the target client. For the names of FrameMakwe clients, see the table below.
`arg`	A string that is passed to the target client.

Returns

`FE_Success` if it succeeds, or the value specified by the target client's last call to `F_ApiReturnValue()`. Note that calls to the structure generator always return `FE_Success` no matter what string is passed to it as an argument.

If `F_ApiCallClient()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_NameNotFound`	There is no client with the specified name in the current FrameMaker product session
`FE_BadParameter`	For the TableSort client only: One of the arguments is invalid. For example, you gave a value for the sort key that is greater than the number of columns or rows in the current table selection, or you have no table cells selected.

F_ApiCenterOnText()

`F_ApiCenterOnText()` centers a range of text so the middle of the text appears in the middle of the document window.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiCenterOnText(F_ObjHandleT docId,
   F_TextRangeT *textRangep);
```

Arguments

`docId`	The ID of the document containing the text range
`textRangep`	The range of text to center

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiCenterOnText()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadRange`	The specified text range is invalid
`FE_NotTextObject`	One of the objects specified for `textRangep` isn't a paragraph (`FO_Pgf`) or a text line (`FO_TextLine`)
`FE_OffsetNotFound`	The offset specified for the text location couldn't be found in the specified paragraph or text line

Example

The following code centers the text selection or insertion point:

```
. . .
F_ObjHandleT docId;
F_TextRangeT tr;

/* Get current text selection */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;
F_ApiCenterOnText(docId, &tr);
. . .
```

See also

"F_ApiScrollToText()" on page 404.

F_ApiCheckStatus()

`F_ApiCheckStatus()` checks the scripts returned by `F_ApiOpen()`, `F_ApiImport()`, `F_ApiSave()`, and `F_ApiUpdateBook()` to determine if a specified status bit is set.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiCheckStatus(F_PropValsT *p,
    IntT statusBit);
```

Arguments

`p`	The property list returned by `F_ApiOpen()`, `F_ApiSave()`, `F_ApiImport()`, or `F_ApiUpdateBook()`
`statusBit`	The status bit to test

Returns

`True` if the bit is set; otherwise, `False`.

Example

The following code determines whether fonts were mapped after a document is opened successfully:

```
. . .
F_PropValsT returnParams;
returnParams = F_ApiAllocatePropVals(FS_NumOpenReturnParams);
. . .
/* Call to F_ApiOpen() goes here. */

if (F_ApiCheckStatus(&returnParams, FV_FontsWereMapped))
    F_ApiAlert("Unavailable fonts were mapped.",
                FF_ALERT_CONTINUE_NOTE);
. . .
```

See also

"F_ApiAddRows()" on page 78, "F_ApiOpen()" on page 355, "F_ApiSave()" on page 399, and "F_ApiUpdateBook()" on page 480.

F_ApiChooseFile()

`F_ApiChooseFile()` displays dialog boxes similar to a Frame product's Open and Save dialog boxes. It displays directories and files in a scroll list and allows the user to choose a file or directory.

Depending on the platform you are running your client on and the constant you specify for `mode`, the dialog box can provide Select, Open, Save, Use, or Cancel buttons. If the user clicks Select, Open, Save, or Use, `F_ApiChooseFile()` stores the selected file or directory's pathname to `*choice`. If the user clicks Cancel, `F_ApiChooseFile()` does not assign a value to `*choice`.

IMPORTANT: `F_ApiChooseFile()` *allocates memory for the string referenced by* `*choice`*. Use* `F_Free()` *to free the string when you are done with it.*

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiChooseFile(StringT *choice,
    StringT title,
    StringT directory,
    StringT stuffVal,
    IntT mode,
    StringT helpLink);
```

Arguments

`choice`	The selected pathname when the user clicks OK.
`title`	The message that appears in the dialog box.
`directory`	The default directory when the dialog box is first displayed. If you specify an empty string, the last directory used by an FDK client is used. If no FDK client has used a directory, the directory specified by the session property, `FP_OpenDir`, is used.
`stuffVal`	The default value that appears in the input field when the dialog box first appears. If the dialog box type specified by `mode` doesn't have an input field, this string is ignored.
`mode`	A constant specifying the type of dialog box. See the table below for possible values.
`helpLink`	Obsolete for versions 6.0 and later; pass an empty string. The name of a document containing help information for the dialog box or an empty string (`""`) if there is no help document.

You can specify the following values for `mode`.

mode constant	Dialog box type
`FV_ChooseSelect`	Dialog box that allows the user to choose a file by clicking Select. It provides an input field that the user can type a filename in.
`FV_ChooseOpen`	Dialog box that allows the user to choose a file by clicking Open. It provides an input field that the user can type a filename in.
`FV_ChooseSave`	Dialog box that allows the user to select a file. It provides Save and Cancel buttons and an input field.
`FV_ChooseOpenDir`	Dialog box that allows the user to choose a directory.

Returns

`0` if the user clicked Open, Select, Use, or Save; a nonzero value if the user clicked Cancel or an error occurred.

If `F_ApiChooseFile()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

The following code displays the dialog boxes shown in Figure 2-3, Figure 2-4, and Figure 2-5

```
. . .
IntT err;
StringT sres;

err = F_ApiChooseFile(&sres, "Choose a file to open.",
                      "", "", FV_ChooseOpen, "");
if (!err) F_ApiDeallocateString(&sres);
err = F_ApiChooseFile(&sres, "Select a file.",
                      "", "", FV_ChooseSelect, "");
if (!err) F_ApiDeallocateString(&sres);
err = F_ApiChooseFile(&sres, "Choose a file to save.", "",
                      "myfile.doc", FV_ChooseSave, "");
if (!err) F_ApiDeallocateString(&sres);
. . .
```

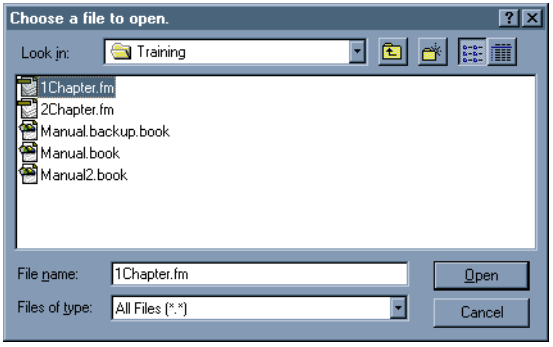


Figure 2-3 *FV_ChooseFile dialog box*

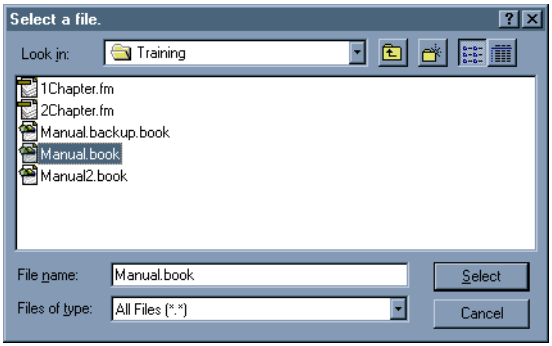


Figure 2-4 *FV_ChooseSelect dialog box*

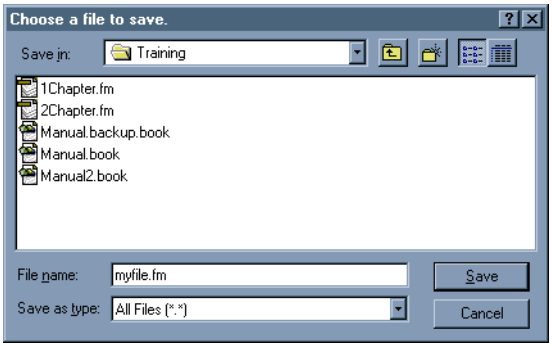


Figure 2-5 *FV_ChooseSave dialog box*

See also

"F_ApiScrollBox()" on page 402.

F_ApiClear()

`F_ApiClear()` deletes the current selection from a document.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiClear(F_ObjHandleT docId
    IntT flags);
```

Arguments

`docId`	The ID of the document from which to clear the selection.
`flags`	Bit field that specifies how to clear the text and how to handle interactive alerts. For default settings, specify `0`.

If you specify `0` for flags, `F_ApiClear()` suppresses any interactive alerts or warnings that arise, leaves the selected table cells empty, and deletes hidden text. You can also OR the following values into `flags`.

flags constant	Meaning
`FF_INTERACTIVE`	Prompt user with dialog or alert boxes that arise
`FF_CUT_TBL_CELLS`	Remove cleared table cells
`FF_VISIBLE_ONLY`	Clear only the visible portion of the selection
`FF_DONT_DELETE_HIDDEN_TEXT`	Don't delete hidden text

The `FF_INTERACTIVE` flag takes precedence over other flags. So, if you specify `FF_INTERACTIVE | FF_DONT_DELETE_HIDDEN_TEXT` and the selection contains hidden text, the Frame product allows the user to choose whether to delete the hidden text.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiClear()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadSelectionForOperation`	Current selection is invalid for this operation
`FE_Canceled`	User or parameters canceled the operation
`FE_WrongProduct`	Current Frame product doesn't support requested operation

Example

The following code extends the text selection to the end of the paragraph, and then deletes it:

```
. . .
F_TextRangeT tr;
F_ObjHandleT docId, pgfId, nextpgfId;

/* Get current text selection. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;
/* Get ID of next paragraph and extend selection to it. */
nextpgfId = F_ApiGetId(docId, tr.end.objId, FP_NextPgfInFlow);
if (nextpgfId)
    {
    tr.end.objId = nextpgfId;
    tr.end.offset = 0;
    F_ApiSetTextRange(FV_SessionId, docId, FP_TextSelection,&tr);
    }

F_ApiClear(docId, 0);
. . .
```

See also

"F_ApiCopy()" on page 115, "F_ApiCut()" on page 123, and "F_ApiPaste()" on page 364.

F_ApiClearAllChangebars()

`F_ApiClearAllChangebars()` clears change bars from a specified document. It executes the same command as clicking the Clear All Change Bars box in the Change Bars dialog box.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiClearAllChangebars(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document for which to clear the change bars

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiClearAllChangebars()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Bad document ID
`FE_WrongProduct`	Current Frame product doesn't support this operation
`FE_SystemError`	System error

Example

The following code removes the change bars from the current document:

```
. . .
F_ObjHandleT docId;
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
F_ApiClearAllChangebars(docId);
. . .
```

F_ApiClientDir()

`F_ApiClientDir()` returns the name of the current FDK client's directory, which is the directory that contains the client's DLL.

Synopsis

```
#include "fapi.h"
. . .
StringT F_ApiClientDir(VoidT);
```

Arguments

None.

Returns

The name of the current FDK client's directory if it succeeds, or `NULL` if an error occurs. If you get a `NULL` string, you should check `FA_errno`; if it is set to `FE_Success`, then there is no `<Directory>` statement in the `apiclients` file. To provide a client directory, use F_ApiSetClientDir(). For more information, see "F_ApiSetClientDir()" on page 412.

If `F_ApiClientDir()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

IMPORTANT: *Use* `F_Free()` *to free the string returned by* `F_ApiClientDir()` *when you are done with it.*

Example

The following code displays the name of the directory containing the current FDK client in an alert box:

```
. . .
#include "fmemory.h"
StringT clientDir;

clientDir = F_ApiClientDir();
F_ApiAlert(clientDir, FF_ALERT_CONTINUE_NOTE);
F_Free(clientDir);
. . .
```

See also

"F_ApiSetClientDir()" on page 412 and "F_ApiClientName()" on page 104.

F_ApiClientName()

`F_ApiClientName()` returns the registered name of the current client (the client that calls `F_ApiClientName()`). For more information on registering FDK clients, see the *FDK Platform Guide* for your platform.

Synopsis

```
#include "fapi.h"
. . .
StringT F_ApiClientName(VoidT);
```

Arguments

None.

Returns

The registered name of the current client if it succeeds, or `NULL` if an error occurs.

If `F_ApiClientName()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

IMPORTANT: *Use* `F_Free()` *to free the string returned by* `F_ApiClientName()` *when you are done with it.*

Example

The following code displays the registered name of the current FDK client in an alert box:

```
. . .
#include "fmemory.h"
StringT clientName;

clientName = F_ApiClientName();
F_ApiAlert(clientName, FF_ALERT_CONTINUE_NOTE);
F_Free(clientName);
. . .
```

See also

"F_ApiClientDir()" on page 103.

F_ApiClientPath()

`F_ApiClientPath()` returns the name of the current FDK client's path, which is the path that contains the client's DLL.

Synopsis

```
#include "fapi.h"
. . .
StringT F_ApiClientPath(ConStringT clientName);
```

Arguments

None.

Returns

The name of the current FDK client's path if it succeeds, or `NULL` if an error occurs..

If `F_ApiClientPath()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

IMPORTANT: *Use* `F_Free()` *to free the string returned by* `F_ApiClientPath()` *when you are done with it.*

Example

The following code displays the name of the path containing the current FDK client in an alert box:

```
. . .
#include "fmemory.h"
StringT clientPath;

clientDir = F_ApiClientPath();
F_ApiAlert(clientDir, FF_ALERT_CONTINUE_NOTE);
F_Free(clientPath);
. . .
```

See also

"F_ApiClientDir()" on page 103 and "F_ApiClientName()" on page 104.

F_ApiClose()

`F_ApiClose()` closes a document, book, dialog box, or Frame session.

If there are unsaved changes in a file and you set `FF_CLOSE_MODIFIED` for the `flags` argument, `F_ApiClose()` abandons the changes and closes the file anyway. If you set `flags` to `0`, `F_ApiClose()` aborts the Close operation and returns `FE_DocModified`.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiClose(F_ObjHandleT Id,
    IntT flags);
```

Arguments

`Id`	The ID of the document, book, dialog box, or session to close.
`flags`	Specifies whether to abort or to close open documents or books if they have unsaved changes. Set the `FF_CLOSE_MODIFIED` flag to close open documents and books regardless of their state.

To quit a Frame session, set `Id` to `FV_SessionId`.

IMPORTANT: *If you are closing an individual document or a dialog box, make sure* `Id` *specifies an object ID and not* `0`*. If* `Id` *is set to* `0`*,* `F_ApiClose()` *quits the Frame session, because the value of the* `FV_SessionId` *constant is* `0`*. If you call* `F_ApiClose()` *in the* `F_ApiDialogEvent()` *callback, call it only to close custom modeless dialog boxes. Calling it for any other reason, such as ending the Frame product session or closing a modal dialog box, will cause unexpected results.*

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiClose()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_DocModified`	The document was modified and `flags` was set to `0`

Example

The following code closes the active document:

```
. . .
F_ObjHandleT docId;
IntT resp;

/* Get ID of active document. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* See whether document has been modified. */
if (F_ApiGetInt(FV_SessionId, docId, FP_DocIsModified))
     resp = F_ApiAlert("Document changed. Close it anyway?",
                       FF_ALERT_OK_DEFAULT);

if (!resp) F_ApiClose (docId, FF_CLOSE_MODIFIED);
. . .
```

See also

"F_ApiSave()" on page 399.

F_ApiCombinedFamilyFonts()

`F_ApiCombinedFamilyFonts()` returns the permutations of angles, variations, and weights available for a specified combined font definition.

IMPORTANT: *To return the permutations of a combined font family, you must use* `F_ApiFamilyFonts()`. *For more information, see "F_ApiFamilyFonts()" on page 165.*

Synopsis

```
#include "fapi.h"
. . .
F_CombinedFontsT F_ApiCombinedFamilyFonts
    (F_ObjHandleT combinedFont);
```

Arguments

`combinedFont`	The object ID of a combined font definition

To get a list of combined font definitions, use `FP_FirstCombinedFontDefnInDoc` to get the first combined font definition in the document. From that you can build a list of combined font definitions using `FP_NextCombinedFontDefnInDoc`.

Returns

An `F_CombinedFontsT` structure provides a list of the permutations of angles, variations, and weights available for the specified combined font definition.

`F_CombinedFontsT` is defined as:

```
typedef struct {
    UIntT len;
    F_CombinedFontT *val;
} F_CombinedFontsT;
```

`F_CombinedFontT` is defined as:

```
typedef struct {
    F_ObjHandelT combinedFont; /* Combined font ID */
    UIntT variation; /* Index of the font variation. */
    UIntT weight; /* Index of the font weight. */
    UIntT angle; /* Index of the font angle. */
} F_CombinedFontT;
```

Example

The following code loops through the combined fonts in a document and prints the combined font name, the base and Western font family names, and the permutations for the combined font:

```
. . .
#include "fapi.h"
#include "futils.h"
#include "fstrings.h"
. . .
UIntT i, base, west;
F_StringsT families, weights, variations, angles;
StringT cFontName;
F_CombinedFontsT perms;
F_ObjHandleT docId, cFontId;
. . .
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
/*Get lists of families, vars, weights, and angles for session*/
families = F_ApiGetStrings(0, FV_SessionId, FP_FontFamilyNames);
weights = F_ApiGetStrings(0, FV_SessionId, FP_FontWeightNames);
variations = F_ApiGetStrings(0, FV_SessionId,
                            FP_FontVariationNames);
angles = F_ApiGetStrings(0, FV_SessionId, FP_FontAngleNames);

/* Loop through combined font definitions for the doc */
cFontId = F_ApiGetId(FV_SessionId, docId,
                     FP_FirstCombinedFontDefnInDoc);
while (cFontId) {
/* Print combined font name, and base and Western family names
*/
    cFontName = F_ApiGetString(docId, cFontId, FP_Name);
    base = F_ApiGetInt(docId, cFontId, FP_BaseFamily);
    west = F_ApiGetInt(docId, cFontId, FP_WesternFamily);
    F_Printf(NULL, (StringT)
        "\n%s is composed of the %s and %s font families.\n",
        cFontName, families.val[base], families.val[west]);
```

```
    /* Print the permutations of the combined font */
    perms = F_ApiCombinedFamilyFonts(cFontId);
    for (i = 0; i < perms.len; i++) {
        F_Printf(NULL, (StringT)
            "Variation: %s, Weight: %s, Angle: %s\n",
            variations.val[perms.val[i].variation],
            weights.val[perms.val[i].weight],
            angles.val[perms.val[i].angle]);
    }
    /* Free the array of F_CombinedFontT structures and */
    /* get the next combined font definition in the document. */
    F_Free(perms.val);
    cFontId = F_ApiGetId(docId, cFontId,
                         FP_NextCombinedFontDefnInDoc);
}
. . .
/* Free the string list structures. */
F_ApiDeallocateStrings(&families);
F_ApiDeallocateStrings(&weights);
F_ApiDeallocateStrings(&variations);
F_ApiDeallocateStrings(&angles);
```

F_ApiCommand()

`F_ApiCommand()` is a callback that you must define to respond to the user choosing a menu item added by your client.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiCommand(command)
IntT command;
{
       /* Code to respond to menu item choices goes here. */
}
```

Arguments

`command`	The `cmd` parameter from the `F_ApiDefineCommand()`, `F_ApiDefineCommandEx()` or `F_ApiDefineAndAddCommand()` call that created the menu item that the user chose

Returns

`VoidT`.

Example

See "Responding to the user choosing a command" on page 209 of the *FDK Programmer's Guide*.

F_ApiCompare()

`F_ApiCompare()` compares two documents or two books.

Synopsis

```
#include "fapi.h"
. . .
F_CompareRetT F_ApiCompare(F_ObjHandleT olderId,
       F_ObjHandleT newerId,
       IntT flags,
       StringT insertCondTag,
       StringT deleteCondTag,
       StringT replaceText,
       IntT compareThreshold);
```

Arguments

`olderId`	The ID of the older version of the document or book.
`newerId`	The ID of the newer version of the document or book.
`flags`	Bit flags that specify how to generate the summary and composite documents. Specify 0 for the default flags.
`insertCondTag`	The condition tag to apply to insertions shown in the composite document. For no insert condition tag, specify `NULL`.
`deleteCondTag`	The condition tag to apply to deletions shown in the composite document. For no delete condition tag, specify `NULL`.
`replaceText`	Text to appear in place of the deleted text. For no replacement text, specify `NULL`.
`compareThreshold`	Percentage of words that can change before paragraphs are considered *not equal*. If two paragraphs are equal, word differences between them are shown within a paragraph in the composite document. If a paragraph is not equal to another, it is marked inserted or deleted. To specify an 85% threshold, set `compareThreshold` to `85`. The default value is `75`.

You can OR the values shown in the following table into the `flags` argument.

flags constant	**Meaning**
`FF_CMP_SUMMARY_ONLY`	Generate a summary document, but not a composite document
`FF_CMP_CHANGE_BARS`	Turn on change bars in the composite document
`FF_CMP_HYPERLINKS`	Put hypertext links in the summary document
`FF_CMP_SUMKIT`	Open the summary document
`FF_CMP_COMPKIT`	Open the composite document

Returns

An `F_CompareRetT` data structure. `F_CompareRetT` is defined as:

```
typedef struct {
   F_ObjHandleT sumId; /* The ID of the summary document */
   F_ObjHandleT compId; /* The ID of the composite document */
} F_CompareRetT;
```

If you use `F_ApiCompare()` to compare two books, it sets `compId` to `0`.

If `F_ApiCompare()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID.
`FE_BadCompare`	`olderId` and `newerId` don't specify the same types of files.
`FE_CompareTypes`	One of the files isn't a FrameMaker product document or book or one file is a book and the other is a document.
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation.

Example

The following code opens two documents and compares them:

```
. . .
F_ObjHandleT oldId, newId;
IntT flags;
F_CompareRetT cmp;

oldId = F_ApiSimpleOpen("/tmp/old.doc", False);
newId = F_ApiSimpleOpen("/tmp/new.doc", False);

flags = FF_CMP_CHANGE_BARS | FF_CMP_COMPKIT;

cmp = F_ApiCompare(oldId, newId, flags, "Comment",
       "", "REPLACED TEXT", 75);

if (!cmp.compId)
   F_ApiAlert("Couldn't compare", FF_ALERT_CONTINUE_NOTE);
. . .
```

F_ApiConnectToSession()

`F_ApiConnectToSession()` connects the calling process to an existing FrameMaker product session process.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiConnectToSession(ConStringT clientName,
   ConStringT hostname,
   IntT prognum);
```

Arguments

`clientName`	The registered name of the client
`hostname`	The host running the FrameMaker product session
`prognum`	The RPC number of the FrameMaker product session

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiConnectToSession()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_DupName`	The client attempted to connect to the same session
`FE_ReadOnly`	The client attempted to connect to a session started with the `-noapi` option
`FE_OutOfRange`	The client attempted to connect to a session of a FrameMaker product version that doesn't support this function
`FE_NameNotFound`	There is no FrameMaker product session

See also

"F_ApiDisconnectFromSession()" on page 159

F_ApiCopy()

`F_ApiCopy()` copies the current selection to the FrameMaker product Clipboard.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiCopy(F_ObjHandleT docId,
    IntT flags);
```

Arguments

`docId`	The ID of the document from which to copy the selection.
`flags`	Bit field that specifies how to copy the text and how to handle interactive alerts. For default settings, specify `0`.

If you specify `0` for flags, `F_ApiCopy()` suppresses any interactive alerts or warnings that arise. You can also OR the following values into `flags`.

flags constant	Meaning
`FF_INTERACTIVE`	Prompt user with dialog or alert boxes that arise
`FF_STRIP_HYPERTEXT`	Do not copy any hypertext markers in the selection
`FF_VISIBLE_ONLY`	Copy only the visible portion of the selection

The `FF_INTERACTIVE` flag takes precedence over other flags. So, if you specify `FF_INTERACTIVE | FF_VISIBLE_ONLY` and the selection is not visible, the FrameMaker product allows the user to choose whether to copy the selection.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiCopy()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_WrongProduct`	Current FrameMaker product doesn't support operation
`FE_BadSelectionForOperation`	Selection doesn't support operation
`FE_Canceled`	User or parameters canceled the operation
`FE_BadOperation`	Parameters specified an invalid operation

Example

The following code selects the first 10 characters of the paragraph containing the insertion point and copies them to the FrameMaker product Clipboard:

```
. . .
F_TextRangeT tr;
F_ObjHandleT docId, pgfId, nextpgfId;

/* Get current text selection. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId,docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Set text selection. */
tr.beg.offset = 0;
tr.end.offset = 10;
F_ApiSetTextRange(FV_SessionId,docId, FP_TextSelection, &tr);

F_ApiCopy(docId, 0);
. . .
```

See also

"F_ApiClear()" on page 100, "F_ApiCut()" on page 123, and "F_ApiPaste()" on page 364.

F_ApiCopy*StructureType*()

The following functions copy frequently used API structures. These functions perform a complete copy, copying any arrays or strings referenced by the structures.

Synopsis

```
#include "fapi.h"
. . .
F_AttributeT F_ApiCopyAttribute(F_AttributeT *fromattribute);
F_AttributeDefT F_ApiCopyAttributeDef(
   F_AttributeDefT *fromattributedef);
F_AttributeDefsT F_ApiCopyAttributeDefs(
   F_AttributeDefsT *fromattributedefs);
F_AttributesT F_ApiCopyAttributes(
   F_AttributesT *fromattributes);
F_ElementCatalogEntriesT F_ApiCopyElementCatalogEntries(
   F_ElementCatalogEntriesT *fromelementcatents);
F_FontsT F_ApiCopyFonts(F_FontsT *fonts);
F_IntsT F_ApiCopyInts(F_IntsT *fromints);
F_MetricsT F_ApiCopyMetrics(F_MetricsT *frommetrics);
F_PointsT F_ApiCopyPoints(F_PointsT *frompoints);
F_PropValT F_ApiCopyProp(F_PropValT *frompropp);
F_PropValsT F_ApiCopyPropVals(F_PropValsT *frompvp);
StringT F_ApiCopyString(ConStringT s);
F_StringsT F_ApiCopyStrings(F_StringsT *fromstrings);
F_TabT F_ApiCopyTab(F_TabT *fromtab);
F_TabsT F_ApiCopyTabs(F_TabsT *fromtabs);
F_TextItemT F_ApiCopyTextItem(F_TextItemT *fromtip);
F_TextItemsT F_ApiCopyTextItems(F_TextItemsT *fromip);
F_UBytesT F_ApiCopyUBytes(F_UBytesT *fromubytes);
F_UIntsT F_ApiCopyUInts(F_UIntsT *fromuints);
F_TypedValT F_ApiCopyVal(F_TypedValT *fromvalp);
```

Arguments

from*StructureType*	The structure to copy

Returns

A copy of the specified structure.

Example

The following code copies the list of font family names available in the current session:

```
. . .
F_StringsT familyNames, strs;

familyNames = F_ApiGetStrings(0, FV_SessionId,
                FP_FontFamilyNames);
strs = F_ApiCopyStrings(&familyNames);

F_ApiDeallocateStrings(&familyNames);
. . .
```

F_ApiCustomDoc()

`F_ApiCustomDoc()` creates a new custom document using the FrameMaker product's default new document template. For more information on default new document templates, see "Documents" on page 73 of the *FDK Programmer's Guide*.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiCustomDoc(MetricT width,
   MetricT height,
   IntT numCols,
   MetricT columnGap,
   MetricT topMargin,
   MetricT botMargin,
   MetricT leftinsideMargin,
   MetricT rightoutsideMargin,
   IntT sidedness,
   BoolT makeVisible);
```

Arguments

`width`	The document page width.
`height`	The document page height.
`numCols`	The default number of columns.
`columnGap`	The default column spacing.

`topMargin`	The document page top margin.
`botMargin`	The document page bottom margin.
`leftinsideMargin`	The left margin for single-sided documents, or the inside margin for double-sided documents.
`rightoutsideMargin`	The right margin for single-sided documents, or the outside margin for double-sided documents.
`sidedness`	A constant that specifies whether the document is single-sided or double-sided and on which side the document starts. See the following table for the list of constants.
`makeVisible`	Specifies whether the document is visible after it is created. Set to `True` to make the document visible.

The `sidedness` argument can have the values shown in the following table.

sidedness constant	New document page characteristics
`FF_Custom_SingleSided`	Single-sided
`FF_Custom_FirstPageRight`	Double-sided, starting with a right page
`FF_Custom_FirstPageLeft`	Double-sided, starting with a left page

Returns

The ID of the new document if it is successful, or `0` if it fails.

If `F_ApiCustomDoc()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation
`FE_BadParameter`	Parameter has an invalid value

Example

The following code creates a custom document with the properties specified in the dialog box in Figure 2-6:

```
. . .
#define in ((MetricT) 65536*72) /* A Frame metric inch. */
F_ObjHandleT docId;
docId = F_ApiCustomDoc(F_MetricFractMul(in,17,2), 11*in, 1,
             F_MetricFractMul(in,1,4), in, in, in, in,
             FF_Custom_SingleSided, True);
. . .
```

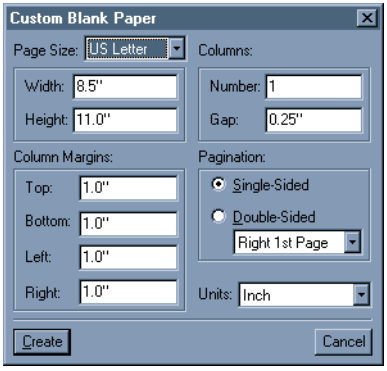


Figure 2-6 *Custom document dialog box*

See also

"F_ApiSimpleNewDoc()" on page 464 and "F_ApiOpen()" on page 355.

F_ApiCut()

`F_ApiCut()` cuts the current selection to the FrameMaker product Clipboard.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiCut(F_ObjHandleT docId,
    IntT flags);
```

Arguments

`docId`	The ID of the document from which to cut the current selection.
`flags`	Bit field that specifies how to cut the text and how to handle interactive alerts. For default settings, specify `0`.

If you specify `0` for flags, `F_ApiCut()` suppresses any interactive alerts or warnings that arise, leaves selected table cells empty, and deletes hidden text. You can also OR the following values into `flags`.

flags constant	Meaning
`FF_INTERACTIVE`	Prompt user with dialog or alert boxes that arise
`FF_CUT_TBL_CELLS`	Remove cut table cells
`FF_VISIBLE_ONLY`	Cut only the visible portion of the selection
`FF_DONT_DELETE_HIDDEN_TEXT`	Don’t cut hidden text

The `FF_INTERACTIVE` flag takes precedence over other flags. So, if you specify `FF_INTERACTIVE | FF_DONT_DELETE_HIDDEN_TEXT` and the selection contains hidden text, the FrameMaker product allows the user to choose whether to delete the hidden text.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs

If `F_ApiCut()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_WrongProduct`	Current FrameMaker product doesn't support requested operation
`FE_BadSelectionForOperation`	Selection doesn't support operation
`FE_Canceled`	User or parameters canceled the operation
`FE_BadOperation`	Parameters specified an invalid operation

Example

The following code cuts the selected text from a document:

```
. . .
F_ApiCut(docId, 0);
. . .
```

See also

"F_ApiClear()" on page 100, "F_ApiCopy()" on page 115, "F_ApiPaste()" on page 364, "F_ApiPopClipboard()" on page 366, and "F_ApiPushClipboard()" on page 384.

F_ApiDeallocate*StructureType*()

The following functions deallocate memory referenced by frequently used API structures. These functions perform a deep deallocation, deallocating any arrays or strings referenced by the structures.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiDeallocateAttribute(F_AttributeT *attribute);
VoidT F_ApiDeallocateAttributeDef(
   F_AttributeDefT *attributedef);
VoidT F_ApiDeallocateAttributeDefs(
   F_AttributeDefsT *attributedefs);
VoidT F_ApiDeallocateAttributes(F_AttributesT *attributes);
VoidT F_ApiDeallocateElementCatalogEntries(
   F_ElementCatalogEntriesT *elementcatents);
VoidT F_ApiDeallocateFonts(F_FontsT *fonts);
VoidT F_ApiDeallocateInts(F_IntsT *ints);
VoidT F_ApiDeallocateMetrics(F_MetricsT *metrics);
VoidT F_ApiDeallocatePoints(F_PointsT *points);
VoidT F_ApiDeallocatePropVal(F_PropValT *prop);
VoidT F_ApiDeallocatePropVals(F_PropValsT *pvp);
VoidT F_ApiDeallocateString(StringT s);
VoidT F_ApiDeallocateStrings(F_StringsT *strings);
VoidT F_ApiDeallocateTab(F_TabT *tab);
VoidT F_ApiDeallocateTabs(F_TabsT *tabs);
VoidT F_ApiDeallocateTextItem(F_TextItemT *tip);
VoidT F_ApiDeallocateTextItems(F_TextItemsT *ip);
VoidT F_ApiDeallocateUBytes(F_UBytesT *ubytes);
VoidT F_ApiDeallocateUInts(F_UIntsT *uints);
VoidT F_ApiDeallocateVal(F_TypedValT *valp);
```

Arguments

StructureType	The structure referencing memory that needs to be deallocated

Returns

VoidT.

Example

The following code deallocates a property list:

```
. . .
F_PropValsT props;
. . .
F_ApiDeallocatePropVals(&props);
. . .
```

F_ApiDefineAndAddCommand()

`F_ApiDefineAndAddCommand()` defines a command (`FO_Command` object) and adds it to a menu or menu bar.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiDefineAndAddCommand(IntT cmd,
    F_ObjHandleT toMenuId,
    StringT name,
    StringT label,
    StringT shortcut);
```

Arguments

`cmd`	The integer that the FrameMaker product passes to your client's `F_ApiCommand()` function when the user chooses the menu item or types the keyboard shortcut for the command. It must be unique for each command in your client, but it need not be unique for different clients.
`toMenuId`	The ID of the menu to which to add the command.
`name`	A unique name for the command. If the user or a client has already defined a command or menu with this name, the new command replaces it.
`label`	The title of the command as it appears on the menu.
`shortcut`	The keyboard shortcut sequence. Many FrameMaker product commands use shortcuts beginning with Escape (\!). To specify Escape when you create a command, use `\\!` in the string you pass to `shortcut`.

To add the new command to a menu that you have created, set `toMenuId` to the ID returned by the `F_ApiDefineMenu()` or `F_ApiDefineAndAddMenu()` call that created the menu.

To add a command to a FrameMaker product menu, you must first get the menu's ID. To get its ID, call `F_ApiGetNamedObject()` with the `objectName` parameter set to its name. For example, to get the ID of the Edit menu, use the following code:

```
. . .
F_ObjHandleT editMenuId;
editMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                        "EditMenu");
. . .
```

The [Files] section of the maker.ini file specifies the location of the menu and command configuration files that list FrameMaker's menus and commands.

The following table lists some FrameMaker product menus and the names you use to specify them:

Menu title	Menu name
Edit	`EditMenu`
File	`FileMenu`
Format	`FormatMenu`
Graphics	`GraphicsMenu`
Special	`SpecialMenu`
Table	`TableMenu`
View	`ViewMenu`
Help	`!HelpMenu`

If you call `F_ApiDefineAndAddCommand()` and specify the name of a command that is already defined in the user's menu configuration files, the FrameMaker product gives precedence to the definition in the configuration files. If the configuration files assign a label or a shortcut to the command, the FrameMaker product uses it instead of the one you specify. If the command is already a menu item, the FrameMaker product ignores the menu that you specify and leaves the menu item where it is.

If the user has already put the menu item specified by `name` on the menus, `F_ApiDefineAndAddCommand()` ignores the `toMenuId` parameter and connects the command to the existing menu item. If the menu item is not already on a menu, `F_ApiDefineAndAddCommand()` adds it to the bottom of the menu specified by `toMenuId`.

IMPORTANT: *If you want to add a command to more than one menu, do not call* `F_ApiDefineAndAddCommand()` *repeatedly to add the command to the menus. This does not work, because if a command already exists,* `F_ApiDefineAndAddCommand()` *ignores the* `toMenuId` *argument and just replaces the existing command with the new command. To add a command to multiple menus, define the command first by calling* `F_ApiDefineCommand()` *or* `F_ApiDefineCommandEx()`*—or call* `F_ApiDefineAndAddCommand()`*, if you want to define and add the command to a menu at the same time—and then call* `F_ApiAddCommandToMenu()` *to add the command to other menus.*

Returns

The ID of the new command (`FO_Command` object), or `0` if an error occurs.

If `F_ApiDefineAndAddCommand()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_BadOperation`	Parameters specify an invalid operation
`FE_NotMenu`	The menu specified by `toMenuId` doesn't exist
`FE_BadParameter`	Parameter has an invalid value
`FE_SystemError`	System error

Example

The following code creates a command named MyCmd and adds it to the Utilities menu:

```
. . .
#define MY_CMD 1
F_ObjHandleT menuId, cmdId;

menuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                         "UtilitiesMenu");
cmdId = F_ApiDefineAndAddCommand(MY_CMD, menuId, "MyCmd",
                     "My Cmd", "\\!MC");
. . .
```

See also

"F_ApiAddCommandToMenu()" on page 72, "F_ApiDefineCommand()" on page 137, and "F_ApiGetNamedObject()" on page 214.

F_ApiDefineAndAddCommandEx()

`F_ApiDefineAndAddCommandEx()` is similar to `F_ApiDefineAndAddCommand()` with an extra parameter to specify the views in which this command is valid.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiDefineAndAddCommandEx (IntT cmd,
F_ObjHandleT toMenuId,
ConStringT tag,
ConStringT label,
ConStringT shortcut,
const F_StringsT *validViewsList);
```

Arguments

`cmd`	The integer that the FrameMaker product passes to the client's `F_ApiCommand()` function when the user chooses the menu item or types the keyboard shortcut for the command. It must be unique for each command in your client, but it need not be unique for different clients.
`toMenuId`	The ID of the menu to which to add the command.
`tag`	A unique name for the command. If the user or a client has already defined a command or menu with this name, the new command replaces it.
`label`	The title of the command as it appears on the menu.
`shortcut`	The keyboard shortcut sequence. Many FrameMaker commands use shortcuts beginning with Escape (\!). To specify Escape when you create a command, use `\\!` in the string you pass to `shortcut`.
`validViewsList`	The view(s) for which this command is valid. The view(s) can be: WYSIWYG View XML View Author View corresponding to the views supported by FrameMaker.

To add the new command to a menu that you have created, set `toMenuId` to the ID returned by the `F_ApiDefineMenu()` or `F_ApiDefineAndAddMenu()` call that created the menu.

To add a command to a FrameMaker product menu, you must first get the menu's ID. To get its ID, call `F_ApiGetNamedObject()` with the `objectName` parameter set to its name. For example, to get the ID of the Edit menu, use the following code:

```
. . .
F_ObjHandleT editMenuId;
editMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                        "EditMenu");
. . .
```

The [Files] section of the maker.ini file specifies the location of the menu and command configuration files that list FrameMaker's menus and commands.

The following table lists some FrameMaker product menus and the names you use to specify them:

Menu title	Menu name
Edit	`EditMenu`
File	`FileMenu`
Format	`FormatMenu`
Graphics	`GraphicsMenu`
Special	`SpecialMenu`
Table	`TableMenu`
View	`ViewMenu`
Help	`!HelpMenu`

If you call `F_ApiDefineAndAddCommandEx()` and specify the name of a command that is already defined in the user's menu configuration files, the FrameMaker product gives precedence to the definition in the configuration files. If the configuration files assign a label or a shortcut to the command, the FrameMaker product uses it instead of the one you specify. If the command is already a menu item, the FrameMaker product ignores the menu that you specify and leaves the menu item where it is.

If the user has already put the menu item specified by `name` on the menus, `F_ApiDefineAndAddCommandEx()` ignores the `toMenuId` parameter and connects the command to the existing menu item. If the menu item is not already on a menu, `F_ApiDefineAndAddCommandEx()` adds it to the bottom of the menu specified by `toMenuId`.

IMPORTANT: *If you want to add a command to more than one menu, do not call* `F_ApiDefineAndAddCommandEx()` *repeatedly to add the command to the menus. This does not work, because if a command already exists,* `F_ApiDefineAndAddCommandEx()` *ignores the* `toMenuId` *argument and just replaces the existing command with the new command. To add a command to multiple menus, define the command first by calling* `F_ApiDefineCommand()` *or* `F_ApiDefineCommandEx()`*—or call* `F_ApiDefineAndAddCommandEx()`*, if you want to define and add the command to a menu at the same time—and then call* `F_ApiAddCommandToMenu()` *to add the command to other menus.*

Returns

The ID of the new command (`FO_Command` object), or `0` if an error occurs.

If `F_ApiDefineAndAddCommandEx()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_BadOperation`	Parameters specify an invalid operation
`FE_NotMenu`	The menu specified by `toMenuId` doesn't exist
`FE_BadParameter`	Parameter has an invalid value
`FE_SystemError`	System error

Example

The following code makes the command valid in WYSIWYG View and Author View:

```
StringT validViews[2] = {(StringT)"WYSIWYG
View",(StringT)"Author View"};
F_StringsT validViewsList = {2, validViews};
F_ObjHandleT cmdId = F_ApiDefineAndAddCommandEx (CMD_NUM,
menuId, "api_command", "API Command", !ac", &validViewsList);
```

The following code makes the command valid in all views:

```
StringT validViews = {(StringT)"All"};
F_StringsT validViewsList = {1, &validViews};
F_ObjHandleT cmdId = F_ApiDefineAndAddCommandEx (CMD_NUM,
menuId, "api_command", "API Command", !ac", &validViewsList)
```

F_ApiDefineAndAddMenu()

`F_ApiDefineAndAddMenu()` defines a menu (`FO_Menu` object) and adds it to another menu.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiDefineAndAddMenu(F_ObjHandleT toMenuId,
    StringT name,
    StringT label);
```

Arguments

`toMenuId`	The name of the menu or menu bar to which to add the new menu.
`name`	A unique name for the new menu. If the user or an FDK client has already defined a command or menu with this name, the new menu replaces it.
`label`	The title of the menu as it appears on the menu bar or menu.

To add a menu to a menu or menu bar created by your client, set `toMenuId` to the ID returned by the `F_ApiDefineMenu()` or `F_ApiDefineAndAddMenu()` call that created the menu or menu bar.

To add a menu to one of a FrameMaker product's menus or menu bars, you must get the menu or menu bar's ID. To get its ID, call `F_ApiGetNamedObject()` with the `objectName` parameter set to its name. For example, to get the ID of the FrameMaker main menu bar, use the following code:

```
. . .
F_ObjHandleT mainMenuId;
mainMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                        "!MakerMainMenu");
. . .
```

The following table lists some of the menu bars that you can add menus to and the strings that specify them. Menu bar names preceded by an exclamation mark (`!`) can't be removed by the user.

FrameMaker product menu bar	toMenuBar string
Menu bar for documents (complete menus)	`!MakerMainMenu`
Menu bar for documents (quick menus)	`!QuickMakerMainMenu`
Menu bar for documents (custom menus)	`!CustomMakerMainMenu`
Menu bar for books (complete menus)	`!BookMainMenu`
Menu bar for books (quick menus)	`!QuickBookMainMenu`
Structure menu bar (structured product interface only)	`!StructureViewMainMenu`
View-only menu bar	`!ViewOnlyMainMenu`

The [Files] section of the maker.ini file specifies the location of the menu and command configuration files that list FrameMaker's menus and commands.

IMPORTANT: *The menu you add appears only on the menu bar you specify. For example, if you add a menu only to the* `!MakerMainMenu` *menu bar, the menu does not appear if the user switches to quick menus. For your menu to appear after the user has switched to quick menus, you must also add it to* `!QuickMakerMainMenu`.

If you call `F_ApiDefineAndAddMenu()` and specify the name of a menu that is already defined in the user's menu configuration files, the FrameMaker product gives precedence to the definition in the configuration files. If the configuration files assign a label to the menu, the FrameMaker product uses it instead of the one you specify. If the menu is already on a menu or menu bar, the FrameMaker product ignores the menu that you specify and leaves the menu where it is.

If the menu specified by `name` is not already on a menu or menu bar, `F_ApiDefineAndAddMenu()` adds it. How the FrameMaker product adds it depends

on the type of menu you are adding it to. The following table lists the types of menus you can add another menu to and how the FrameMaker product adds the menu.

Type of menu or menu bar you are adding a menu to	How the FrameMaker product implements the added menu	Where the FrameMaker product adds the menu
Menu bar	Pull-down menu	At the right of the menu bar
Pull-down menu	Pull-right menu	At the bottom of the pull-down menu
Pop-up menu	Pull-right menu	At the bottom of the pop-up menu
Pull-right menu	Pull-right menu	At the bottom of the pull-right menu

Returns

The ID of the new menu (`FO_Menu` object), or `0` if an error occurs.

If `F_ApiDefineAndAddMenu()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_BadOperation`	Parameters specify an invalid operation
`FE_NotMenu`	The menu specified by `toMenuId` doesn't exist or isn't a menu; `tag` specifies an existing command
`FE_BadParameter`	Parameter has an invalid value
`FE_SystemError`	System error

Example

The following code defines and adds a menu named APIMenu to the FrameMaker main menu:

```
. . .
#define MY_CMD 1
F_ObjHandleT menubarId, menuId, cmdId;

/* Get ID of FrameMaker main menu. */
menubarId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                    "!MakerMainMenu");

/* Define and add a menu to the main menu. */
menuId = F_ApiDefineAndAddMenu(menubarId, "APIMenu",
                    "API Menu");

/* Add a command to the menu. */
cmdId = F_ApiDefineAndAddCommand(MY_CMD, menuId, "MyCmd",
                    "My Cmd", "");
. . .
```

See also

"F_ApiAddMenuToMenu()" on page 74, "F_ApiDefineMenu()" on page 141, and "F_ApiGetNamedObject()" on page 214.

F_ApiDefineCommand()

`F_ApiDefineCommand()` defines a command (`FO_Command` object). After you define a command, you can add it to a menu with `F_ApiAddCommandToMenu()`.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiDefineCommand(IntT cmd,
    StringT name,
    StringT label,
    StringT shortcut);
```

Arguments

`cmd`	The integer that the FrameMaker product passes to your client's `F_ApiCommand()` function when the user executes the command. It must be unique for each menu item in your client, but it need not be unique for different clients.
`name`	A unique name for the command. If the user or a client has already defined a command or menu with this name, the new command replaces it.
`label`	The title of the menu item as it appears on the menu.
`shortcut`	The keyboard shortcut sequence. Many FrameMaker product commands use shortcuts beginning with Escape (\!). To specify Escape when you create a command, use `\\!` in the string you pass to `shortcut`.

If you call `F_ApiDefineCommand()` and specify the name of a command that is already defined in the user's menu configuration files, the FrameMaker product gives precedence to the definition in the configuration files. If the configuration files assign a label or a shortcut to the command, the FrameMaker product uses it instead of the one you specify.

Returns

The ID of the new command (`FO_Command` object), or `0` if an error occurs.

If `F_ApiDefineCommand()` fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_BadOperation`	Parameters specify an invalid operation
`FE_BadParameter`	Parameter has an invalid value
`FE_NotCommand`	`tag` specifies a menu; can't redefine a menu as a command
`FE_SystemError`	System error

Example

The following code defines a command named CheckGrammar that has a keyboard shortcut of Esc C G:

```
. . .
#define CHECK_GRAMMAR 1
F_ObjHandleT cmdId;

cmdId = F_ApiDefineCommand(CHECK_GRAMMAR, "CheckGrammar",
              "Check Grammar", "\\!CG");
. . .
```

See also

"F_ApiDefineCommandEx()" on page 139, "F_ApiAddCommandToMenu()" on page 72, "F_ApiDefineAndAddCommand()" on page 126, and "F_ApiGetNamedObject()" on page 214.

F_ApiDefineCommandEx()

`F_ApiDefineCommandEx()` is similar to `F_ApiDefineCommand()` with an extra parameter to specify views in which this command is valid. After you define a command, you can add it to a menu with `F_ApiAddCommandToMenu()`.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiDefineCommandEx(IntT cmd,
    ConStringT name,
    ConStringT label,
    ConStringT shortcut,
    const F_StringsT *validViewsList);
```

Arguments

`cmd`	The integer that the FrameMaker product passes to your client's `F_ApiCommand()` function when the user executes the command. It must be unique for each menu item in your client, but it need not be unique for different clients.
`name`	A unique name for the command. If the user or a client has already defined a command or menu with this name, the new command replaces it.
`label`	The title of the menu item as it appears on the menu.
`shortcut`	The keyboard shortcut sequence. Many FrameMaker product commands use shortcuts beginning with Escape (\!). To specify Escape when you create a command, use `\\!` in the string you pass to `shortcut`.
`validViewsList`	The views for which this command is valid. The view(s) can be: WYSIWYG View Code View Author View corresponding to the views supported by FrameMaker.

If you call `F_ApiDefineCommandEx()` and specify the name of a command that is already defined in the user's menu configuration files, the FrameMaker product gives precedence to the definition in the configuration files. If the configuration files assign a label or a shortcut to the command, the FrameMaker product uses it instead of the one you specify.

Returns

The ID of the new command (`FO_Command` object), or `0` if an error occurs.

If `F_ApiDefineCommandEx()` fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_BadOperation`	Parameters specify an invalid operation
`FE_BadParameter`	Parameter has an invalid value
`FE_NotCommand`	`tag` specifies a menu; can't redefine a menu as a command
`FE_SystemError`	System error

Example

The following code makes the comman valid in WYSIWYG view and Author view:

```
. . .
StringT validViews[2] = {(StringT)"WYSIWYG iew",(StringT)"Author
View"};
F_StringsT validViewsList = {2, validViews};
F_ObjHandleT cmdId = F_ApiDefineCommandEx (CMD_NUM,
"api_command", "API Command", "!ac", &validViewsList);
. . .
```

See also

"F_ApiAddCommandToMenu()" on page 72, "F_ApiDefineAndAddCommand()" on page 126, and "F_ApiGetNamedObject()" on page 214.

F_ApiDefineMenu()

`F_ApiDefineMenu()` defines a menu (`FO_Menu` object). After you define a menu, you can add it to a menu or a menu bar with `F_ApiAddMenuToMenu()`.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiDefineMenu(StringT name,
    StringT label);
```

Arguments

`name`	A unique name for the menu. If the user or an FDK client has already defined a command or menu with this name, the new menu replaces it.
`label`	The title of the menu as it appears on the menu bar or menu.

If you call `F_ApiDefineMenu()` and specify the name of a menu that is already defined in the user's menu configuration files, the FrameMaker product gives precedence to the definition in the configuration files. If the configuration files assign a label to the menu, the FrameMaker product uses it instead of the one you specify.

If the user has already defined a menu with the name specified by `name`, `F_ApiDefineMenu()` ignores the `label` parameter and uses the label specified by the user.

Returns

The ID of the new menu (`FO_Menu` object), or `0` if an error occurs.

If `F_ApiDefineMenu()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_BadOperation`	Parameters specify an invalid operation
`FE_NotMenu`	`tag` specifies a command; can't redefine a command as a menu
`FE_BadParameter`	Parameter has an invalid value
`FE_SystemError`	System error

Example

The following code adds a menu named Search to the menu bar for books. It also adds a command labeled Find and Replace to the Search menu.

```
. . .
#define FIND_REPLACE 1
F_ObjHandleT menubarId, menuId;

/* Define menu and add a command to it. */
menuId = F_ApiDefineMenu("SearchMenu", "Search");
F_ApiDefineAndAddCommand(FIND_REPLACE, menuId, "FindReplace",
                    "Find and Replace", "\\!fR");
/* Put the menu on the menu bar for books. */
menubarId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                    "!BookMainMenu");
F_ApiAddMenuToMenu(menubarId, menuId);
. . .
```

See also

"F_ApiAddMenuToMenu()" on page 74, "F_ApiDefineAndAddMenu()" on page 134, and "F_ApiGetNamedObject()" on page 214.

F_ApiDelete()

`F_ApiDelete()` deletes an object from a document. Be careful about possible side effects when you use `F_ApiDelete()`. When you delete some objects, the API deletes any objects they contain. For example, when you delete a table, the API deletes all the `FO_Row` and `FO_Cell` objects that it contains. When you delete a frame, the API deletes all the objects in it.

The following types of objects can't be deleted with `F_ApiDelete()`:

- `FO_Book`
- `FO_Doc`
- `FO_HiddenPage`
- `FO_Cell`
- `FO_MasterPage` (Left and Right master pages only)
- `FO_UnanchoredFrame` (page frames only)
- `FO_Color` (predefined colors only)
- `FO_Session`
- `FO_VarFmt` (system variables only)

To delete table rows and columns, use `F_ApiDeleteCols()` and `F_ApiDeleteRows()` instead of `F_ApiDelete()`. To delete a page frame, delete the page that contains it. To delete text, see "F_ApiDeleteText()" on page 151.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiDelete(F_ObjHandleT docId,
       F_ObjHandleT objId);
```

Arguments

`docId`	The ID of the document, book, or menu containing the object
`objId`	The ID of the object to remove

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiDelete()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadDelete`	Specified object couldn't be deleted
`FE_BadOperation`	Function call specified an illegal operation
`FE_BadParameter`	Function call specified an invalid parameter
`FE_NotMenu`	`objId` is a menu item (`FO_Menu`, `FO_Command`, or `FO_MenuItemSeparator`), but `docId` is not the ID of the menu containig it

Example

The following code deletes all the markers in the current document:

```
. . .
F_ObjHandleT docId, mrkrId, prvmrkrId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Get first marker, then traverse markers and delete them. */
mrkrId = F_ApiGetId(FV_SessionId, docId, FP_FirstMarkerInDoc);
while (mrkrId)
    {
    /* As each marker is deleted, its FP_NextMarkerInDoc property
    ** becomes invalid, so it is necessary to get the next marker
    ** before deleting the current one.
    */
    prvmrkrId = mrkrId;
    mrkrId = F_ApiGetId(docId, prvmrkrId, FP_NextMarkerInDoc);
    F_ApiDelete(docId, prvmrkrId);
    }
. . .
```

F_ApiDeleteAllKeyDefinitions()

Deletes all the key definitions in the specified key catalog.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiDeleteAllKeyDefinitions(F_ObjHandleT keyCatalogId)
```

Arguments

`keyCatalogId`	The ID of the key catalog from which to delete all key definitions.

Returns

If `F_ApiDeleteAllKeyDefinitions()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_BadObjId`	The ID provided does not specify a key catalog.

F_ApiDeleteCols()

`F_ApiDeleteCols()` deletes columns from a table. To delete an entire table, use `F_ApiDelete()`.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiDeleteCols(F_ObjHandleT docId,
    F_ObjHandleT tableId,
    IntT delColNum,
    IntT numDelCols);
```

Arguments

`docId`	The ID of the document containing the table.
`tableId`	The ID of the table containing the columns.
`delColNum`	The first column to delete. Columns are numbered from left to right, starting with `0`.
`numDelCols`	The number of columns to delete.

`F_ApiDeleteCols()` deletes the column specified by `delColNum` and `(numDelCols-1)` columns to the right of it.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiDeleteCols()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support requested operation
`FE_BadOperation`	Parameter specifies an invalid operation
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid table ID

Example

The following code deletes the first two columns in the selected table in the active document:

```
. . .
F_ObjHandleT docId, tblId;

/* Get the document and table IDs. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tblId = F_ApiGetId(FV_SessionId, docId, FP_SelectedTbl);

F_ApiDeleteCols(docId, tblId, 0, 2);
. . .
```

See also

"F_ApiDelete()" on page 143 and "F_ApiDeleteRows()" on page 148.

Structured F_ApiDeleteCondTag()

`F_ApiDeleteCondTag()` deletes a conditional tag from a document.

Synopsis

```
#include "fapi.h"

StatusT F_ApiDeleteCondTag(F_ObjHandleT docId, F_ObjHandleT
condTagId, IntT action);
```

Arguments

`docId`	The ID of the document containing the conditional tag to be deleted
`condTagId`	The ID of the conditional tag object (type: `FO_CondFmt`) in the doc
`action`	One of the following: `FF_UNTAGGED_ASK`: Prompt the user `FF_UNTAGGED_UNCOND`: Make text unconditional `FF_UNTAGGED_DELETE`: Delete text

Returns

Returns one of the following

`FE_BadDocId`	Document id is invalid
`FE_BadObjId`	Conditional tag object id is invalid.
`FE_ReadOnly`	Document is read only.
`FE_BadValue`	Action is not one of the specified values
`FE_Success`	Deletion was successful

F_ApiDeletePropByName()

`F_ApiDeletePropByName()` deletes an inset facet.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiDeletePropByName(F_ObjHandleT docId,
    F_ObjHandleT objId,
    StringT propName);
```

Arguments

`docId`	The ID of the document containing the inset whose facet you want to delete
`objId`	The ID of the inset
`propName`	The name of the facet you want to delete

Returns

`VoidT`.

If `F_ApiDeletePropByName()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadPropNum`	Specified property number is invalid
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropType`	Incorrect property type for this function

Example

The following code deletes a facet called `wks.facet`:

```
. . .
F_ObjHandleT docId, insetId;

F_ApiDeletePropByName(docId, insetId, "wks.facet");
. . .
```

F_ApiDeleteRows()

`F_ApiDeleteRows()` deletes rows from a table. Like the Delete command in the FrameMaker product user interface, `F_ApiDeleteRows()` does not allow you to delete more than one type of row at time. The range of rows you specify must be all body rows, all header rows, or all footer rows.

To delete an entire table, use `F_ApiDelete()`.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiDeleteRows(F_ObjHandleT docId,
    F_ObjHandleT tableId,
    F_ObjHandleT refRowId,
    IntT numDelRows);
```

Arguments

`docId`	The ID of the document containing the table
`tableId`	The ID of the table containing the rows
`refRowId`	The ID of the first row to delete
`numDelRows`	The number of rows to delete, including `refRowId`

`F_ApiDeleteRows()` deletes `refRowId` and (`numDelRows - 1`) rows below it.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiDeleteRows()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support the requested operation.
`FE_BadOperation`	Parameter specifies an invalid operation.
`FE_BadDocId`	Invalid document ID.
`FE_BadObjId`	Invalid table or row ID.
`FE_OutOfRange`	The `refRowId` parameter does not specify a row in the table, or the specified range includes more than one type of row (for example, header rows and body rows).

Example

The following code deletes the first row of the selected table:

```
. . .
F_ObjHandleT docId, tblId, rowId;

/* Get the document and selected table IDs. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tblId = F_ApiGetId(FV_SessionId, docId, FP_SelectedTbl);

/* Get the ID for the first row, then delete it. */
rowId = F_ApiGetId(docId, tblId, FP_FirstRowInTbl);

F_ApiDeleteRows(docId, tblId, rowId, 1);
. . .
```

See also

"F_ApiDelete()" on page 143 and "F_ApiDeleteCols()" on page 145.

F_ApiDeleteText()

`F_ApiDeleteText()` deletes a specified text range from a document.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiDeleteText(F_ObjHandleT docId,
   F_TextRangeT *textRangep);
```

Arguments

`docId`	The ID of the document from which to delete the text
`textRangep`	The text range to delete

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiDeleteText()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDelete`	Specified text couldn't be deleted
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadRange`	Specified text range is invalid
`FE_NotTextObject`	Object specified for the text range isn't an object that contains text, for example, a text frame (`FO_TextFrame`), a paragraph (`FO_Pgf`) or a text line (`FO_TextLine`)
`FE_OffsetNotFound`	Offset specified for the text range couldn't be found in the specified paragraph or text line
`FE_BadSelectionForOperation`	Selection is within a locked text range.

Example

The following code gets the text selection from the active document and deletes it:

```
. . .
F_TextRangeT tr;
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId,docId, FP_TextSelection);
if(tr.beg.objId == 0) return;
F_ApiDeleteText(docId, &tr);
. . .
```

See also

"F_ApiClear()" on page 100 and "F_ApiAddText()" on page 80.

F_ApiDeleteTextInsetContents()

`F_ApiDeleteTextInsetContents()` deletes the text in a text inset. You must unlock a text inset before you call `F_ApiDeleteTextInsetContents()` to delete its contents. After you are done, you must relock the text inset.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiDeleteTextInsetContents(F_ObjHandleT docId,
   F_ObjHandleT insetId);
```

Arguments

`docId`	The ID of the document containing the inset
`insetId`	The text inset containing the text to be deleted

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiDeleteTextInsetContents()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDelete`	Specified text couldn't be deleted
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	The ID does not specify a text inset
`FE_BadSelectionForOperation`	The specified text inset is locked

Example

See "Updating a client text inset" on page 469 of the *FDK Programmer's Guide*.

See also

"F_ApiDelete()" on page 143.

Structured F_ApiDeleteUndefinedAttribute()

`F_ApiDeleteUndefinedAttribute()` deletes from structural elements, an attribute for which no value is assigned. You can delete undefined attributes for a given element, all elements of a specific type, or all elements in the document.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiDeleteUndefinedAttribute(F_ObjHandleT docId,
       StringT attrName, IntT scope, F_ObjHandleT objId);
```

Arguments

`docId`	The ID of the document containing the element or elements whose attributes you want to delete
`attrName`	The name of the attribute you want to delete
`scope`	One of `FV_Element`, `FV_ElementsOfType`, or `FV_AllElements`
`objId`	The element or element definition from which to delete the undefined attributes

The value for `objId` indicates a different type of object, depending on the value of `scope`, as indicated by the following table:

If scope is:	objId must be:
FV_Element	The ID of the element from which you want to delete the undefined attribute
FV_ElementsOfType	The element definition for the type of element from which you want to delete the undefined attributes
FV_AllElements	0

Returns

FE_Success if it succeeds, or an error code if an error occurs.

If F_ApiDeleteUndefinedAttribute() fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
FE_BadDocId	Invalid document ID
FE_BadObjId	Invalid object ID
FE_AttributeNotUndefined	Specified attribute in specified element contains a value

Example

The following code deletes all undefined instances of the attribute named “author” in the document:

```
. . .
F_ObjHandleT docId;
ErrorT err;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);


err = F_ApiDeleteUndefinedAttribute(docId, (StringT)"author",
          FV_AllElements, 0);
. . .
```

Structured F_ApiDemoteElement()

F_ApiDemoteElement() demotes the selected structural element or elements. The element becomes a child of the sibling element before it.

IMPORTANT: *At least one structural element must be selected when* `F_ApiDemoteElement()` *is called.*

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiDemoteElement(F_ObjHandleT docId);
```

Arguments

`docId` The document that contains the element selection

Returns

`VoidT`.

If `F_ApiDemoteElement()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product isn't Structured FrameMaker
`FE_BadDocId`	Invalid document ID
`FE_BadSelectionForOperation`	Current text selection is invalid for this operation

Example

The following code demotes the selected elements in the active document:

```
. . .
F_ApiDemoteElement(F_ApiGetId(0, FV_SessionId, FP_ActiveDoc));
. . .
```

See also

"F_ApiPromoteElement()" on page 377.

F_ApiDialogEvent()

`F_ApiDialogEvent()` is a callback that you can define in your client. The FrameMaker product calls `F_ApiDialogEvent()` when an event occurs in a dialog box opened by your client.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiDialogEvent(IntT dlgNum,
    IntT itemNum,
    IntT modifiers);
```

Arguments

`dlgNum`	The number of the dialog in which the event occurred (the number you specified when you called `F_ApiModalDialog()` or `F_ApiModelessDialog()` to display the dialog)
`itemNum`	If an event occurred in a specific dialog item, the number (a nonnegative integer) of the dialog item. If the event doesn't apply to a specific dialog item, a negative integer constant specifying the event. See the table below for a list of constants.
`modifiers`	Bit flags specifying which modifier keys the user was holding down when the event occurred. See the table below for a list of possible flags.

The API can set the `itemNum` parameter function to the ID of a dialog item or to one of the following negative constants:

Constant	Meaning
`FV_DlgClose`	The user closed the dialog box.
`FV_DlgEnter`	The user moved input focus to the dialog box.
`FV_DlgNoChange`	The user pressed Shift-F8 to set the dialog box to As Is settings.

Constant	Meaning
`FV_DlgReset`	The user pressed Shift-F9 to reset the dialog box.

`FV_DlgResize`	A modeless dialog was resized. Here is a usage example:
	`VoidT F_ApiDialogEvent(IntT dlgNum, IntT itemNum, IntT modifiers)`
	`{`
	`switch(dlgNum)`
	`{`
	`case DLG_NUM1:`
	`{`
	`If (itemNum == FV_DlgResize)`
	`{`
	`/* 'dlgNum' dialog has been resized - do something if needed */`
	`}`
	`}`
	`break;`
	`}`
	`}`

The `modifiers` parameter can have the following bit flags set:

Flag	Meaning
`FV_EvCaps`	The Caps Lock key is on
`FV_EvControl`	The user was holding down the Control key
`FV_EvMeta`	The user was holding down the the Alt key (on Windows)
`FV_EvShift`	The user was holding down the Shift key

Returns

`VoidT`.

Example

See “Handling user actions in dialog boxes” on page 448 of the *FDK Programmer’s Guide*.

See also

“F_ApiDialogItemId()” on page 158.

F_ApiDialogItemId()

`F_ApiDialogItemId()` returns the ID of a dialog item with a specified item number.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiDialogItemId(F_ObjHandleT dialogId,
    IntT itemNum);
```

Arguments

`dialogId`	The ID of the dialog box containing the item
`itemNum`	The item number of the item

Returns

The ID of the item with the specified number or `0` if there is no item with the specified number.

Example

The following code gets the ID of the dialog with the number `APPLY_BUTTON`.

```
. . .
#define APPLY_BUTTON 1
F_ObjHandleT dlgId, dlgItemId;
. . .
dlgItemId = F_ApiDialogItemId(dlgId, APPLY_BUTTON);
. . .
```

See also

“F_ApiDialogEvent()” on page 155.

F_ApiDisconnectFromSession()

`F_ApiDisconnectFromSession()` ends communication with a FrameMaker product process.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiDisconnectFromSession();
```

Arguments

None.

Returns

`FE_Success` if it succeeds, or a system error code if an error occurs.

See also

"" on page 173, "F_ApiConnectToSession()" on page 115

Structured F_ApiElementDefIsText()

Some structural elements in FrameMaker documents are placeholders for text. For example, when a Para element contains text with a cross-reference element embedded in it, the ranges of text that surround the cross-reference element are treated as elements themselves. These elements are called *text nodes*.

When you are traversing the elements in a container, it is often useful to know if a given element is a text node. `F_ApiElementDefIsText()` checks the value of an element definition to determine whether the element it is applied to is a text node.

`F_ApiElementDefIsText()` is implemented as a macro.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiElementDefIsText(F_ObjHandleT docId,
                F_ObjHandleT objId);
```

Arguments

`docId`	The ID of the document containing the element definition
`objId`	The ID of the element definition (`FO_ElementDef`)

Returns

`True` if the element definition corresponds to that of a text node, or `False` if it doesn't.

Example

The following code displays an alert if an element is a text node:

```
. . .
F_ObjHandleT edefId, objId;
. . .
if (F_ApiElementDefIsText(docId, edefId))
   F_ApiAlert("Element is text node.", FF_ALERT_CONTINUE_NOTE);
. . .
```

Structured F_ApiElementLocToTextLoc()

`F_ApiElementLocToTextLoc()` returns the text location structure that corresponds with the current element location.

Synopsis

```
#include "fapi.h"
. . .
F_TextLocT F_ApiElementLocToTextLoc (F_ObjHandleT docId,
                 const F_ElementLocT *elocp);
```

Arguments

`docId`	The ID of the document containing the element
`elocp`	The element location structure to convert

Returns

An `F_TextLocT` structure specifying a text location. The `F_TextLocT` structure is defined as:

```
typedef struct{
    F_ObjHandleT objId; /* Object containing text */
    IntT offset; /* Characters from start of object */
} F_TextLocT;
```

If `F_ApiElementLocToTextLoc()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadParameter`	`elocp` was empty or a parameter was improperly specified
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code converts an element location to a text location and gets the name of the paragraph format for the text location:

```
. . .
F_ObjHandleT docId;
F_ElementRangeT eRange;
F_ElementLocT eloc;
F_TextLocT textLoc;
StringT paraTag;
. . .
eRange = F_ApiGetElementRange(FV_SessionId, docId,
                        FP_ElementSelection);
eloc = eRange.beg;
textLoc = F_ApiElementLocToTextLoc (docId, &eloc);
if (F_ApiGetObjectType(docId, textLoc.objId) == FO_Pgf)
   paraTag = F_ApiGetString(docId, textLoc.objId, FP_Name);
. . .
```

See also

"F_ApiTextLocToElementLoc()" on page 473.

F_ApiEnableUnicode()

Used to enable *Unicode Mode* or *Compatibility Mode.*

Synopsis

```
#include "fapi.h"
...
VoidT F_ApiEnableUnicode(BoolT enable);
```

Arguments

`enable`	`True` enables *Unicode Mode*, `False` enables *Compatibility Mode*. Default: `False`

Returns

`VoidT`

Example

The following example creates two alerts.

The first one shows the string "`This will be treated as FrameRoman on English locale: Ð§ ÎÆ§ ÿ¥`" on an English locale.

The second one shows the string "`This will be treated as Unicode on every locale:` ä 뮤 ش ." on any locale.

```
#include "fencode.h"
. . .
F_ApiAlert("This will be treated as FrameRoman on English
locale:
\xC3\xA4 \xEB\xAE\xA4 \xD8\xB4",
   FF_ALERT_CONTINUE_NOTE);

F_ApiEnableUnicode(True);
F_ApiAlert("This will be treated as Unicode on every locale:
\xC3\xA4
\xEB\xAE\xA4 \xD8\xB4",
   FF_ALERT_CONTINUE_NOTE);
. . .
```

F_ApiErr()

`F_ApiErr()` prints your client's name and a message to the console.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiErr(StringT message);
```

Arguments

`message` The message to print

Returns

`VoidT`.

Example

The following code prints the message `Frame API client "myclient": There's been a problem with this client.` when a problem occurs with the client:

```
. . .
F_ApiErr("There's been a problem with this client.\n");
. . .
```

F_ApiFamilyFonts()

F_ApiFamilyFonts() returns the permutations of angles, variations, and weights available for a specified font family.

IMPORTANT: *To return the permutations of a combined font, you must use* F_ApiCombinedFamilyFonts(). *For more information, see* *"F_ApiCombinedFamilyFonts()" on page 109.*

Synopsis

```
#include "fapi.h"
. . .
F_FontsT F_ApiFamilyFonts(IntT family);
```

Arguments

family	The index of the font family (in the list of fonts in the session)

To get the list of font families, angles, variations, or weights in the current FrameMaker product session, use F_ApiGetStrings(). For more information, see "F_ApiGetStrings()" on page 242.

Returns

An F_FontsT structure providing a list of the permutations of angles, variations, and weights available for the specified font family.

F_FontsT is defined as:

```
typedef struct {
   UIntT len; /* The number of structures referenced by val.*/
   F_FontT *val; /* Structures listing the permutations. */
} F_FontsT;
```

F_FontT is defined as:

```
typedef struct {
   UIntT family; /* The index of the font family name. */
   UIntT variation; /* The index of the variation name. */
   UIntT weight; /* The index of the weight name. */
   UIntT angle; /* The index of the angle name. */
} F_FontT;
```

Example

The following code prints the variations, weights, and angles available for Helvetica in the current session:

```
. . .
#include "futils.h"
#include "fstrings.h"
. . .
UIntT i, j;
F_StringsT families, weights, variations, angles;
F_FontsT perms;

/* Get lists of families, variations, weights, and angles. */
families = F_ApiGetStrings(0, FV_SessionId, FP_FontFamilyNames);
weights = F_ApiGetStrings(0, FV_SessionId, FP_FontWeightNames);
variations = F_ApiGetStrings(0, FV_SessionId,
                FP_FontVariationNames);
angles = F_ApiGetStrings(0, FV_SessionId, FP_FontAngleNames);

/* Get the index for Helvetica. */
for (i=0; i < families.len; i++)
   if(F_StrIEqual(families.val[i], "helvetica")) break;
if (i == families.len) return; /* Helvetica not found. */

/* Get the permutations and print them to the console. */
perms = F_ApiFamilyFonts(i);
for (j=0; j < perms.len; j++)
{
   F_Printf(NULL, "Variation: %s, Weight: %s, Angle: %s\n",
      weights.val[perms.val[j].variation],
      variations.val[perms.val[j].weight],
      angles.val[perms.val[j].angle]);
}

/* Free the structures and strings. */
F_ApiDeallocateFonts(&perms);
F_ApiDeallocateStrings(&families);
F_ApiDeallocateStrings(&weights);
F_ApiDeallocateStrings(&variations);
F_ApiDeallocateStrings(&angles);
. . .
```

F_ApiFcodes()

`F_ApiFcodes()` sends an array of function codes (f-codes) to the FrameMaker product. *F-codes* are hexadecimal codes that specify individual user actions, such as cursor movement and text entry. When you use `F_ApiFcodes()` to send an array of f-codes to the FrameMaker product, it executes each f-code as if the user performed the action.

`F_ApiFcodes()` uses the current focus in a dialog box or a visible document. If you want to execute a set of f-codes in a particular dialog box or document, make sure that the dialog box or document is active (or has focus).

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiFcodes(IntT len,
    IntT* vec);
```

Arguments

`len`	The length of the array of f-codes
`vec`	The array of f-codes to send to the FrameMaker product

For a complete list of f-codes, see the `fcodes.h` file shipped with the FDK. To use FDK-defined f-code constants, such as `KBD_RETURN`, include this file in your client.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiFcodes()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

The following code executes f-codes to move the cursor down one line and move the current document window to the back:

```
. . .
#include "fm_commands.h"
. . .
static IntT fcodes[] = {CSR_DOWN, KBD_HIDEWIN};
F_ApiFcodes(sizeof(fcodes)/sizeof(IntT), fcodes);
. . .
```

F_ApiFind()

`F_ApiFind()` performs the same actions as using the Find dialog box to search a document for text or other types of content.

Synopsis

```
#include "fapi.h"
. . .
F_TextRangeT F_ApiFind(F_ObjHandleT docId,
    const F_TextLocT *textLocp,
    const F_PropValsT *findParamsp);
```

Arguments

`docId`	The ID of the document to search.
`textLocp`	The text location to begin searching from.
`findParamsp`	A property list that specifies what to search for.

The `findParamsp` parameter points to a property list that contains:

- FS_FindCustomizationFlags, an optional property you can use to customize the search.
- One of a list of a list of properties that specify what type of content to search for; text, elements, character formats, etc. You must specify one of these properties.

The properties you can assign to findParamsp are:

Property	Meaning and possible values
FS_FindCustomizationFlags	An optional parameter of type FT_Integer that may be any of the following bit flags OR'ed together: FF_FIND_CONSIDER_CASE, FF_FIND_WHOLE_WORD, FF_FIND_USE_WILDCARDS, FF_FIND_BACKWARDS If no customization flags are specified the default is to search forward, to not use wildcards, to not consider case, and to not use whole words.
FS_FindWrap	A BoolT flag that determines whether the find operation will wrap when it reaches the location where the search began. Default is True; the find operations wraps. If False, after reaching the location where the search began, the find operation returns an empty F_TextRangeT and FA_errno is set to FE_NotFound.
FS_FindText	FT_String search text

FS_FindElementTag	FT_Strings as follows:
	propVal.u.ssval.len = FV_NumFindElementItems;
	propVal.u.ssval.val[FV_FindElemTag] = [an_element_tag];
	propVal.u.ssval.val[FV_FindAttrName]= [an_attribute_name];
	propVal.u.ssval.val[FV_FindAttrValue] = [an_attribute_value];
	All of the strings must be present, but any or all may be empty

Property	Meaning and possible values
FS_FindCharFmt	No associated property[a]. One or more of the following additional properties should be specified to tailor the search FP_FontFamily FP_CombinedFont FP_FontSize FP_FontAngle FP_FontWeight FP_FontVariation FP_Color FP_Spread FP_Stretch FP_Language FP_Underline FP_Overline FP_Strikethrough FP_ChangeBar FP_Capitalization FP_Position FP_Tsume[b]
FS_FindPgfTag	FT_String paragraph tag
FS_FindCharTag	FT_String character tag
FS_FindTableTag	FT_String table tag
FS_FindObject	FV_FindAnyMarker FV_FindAnyXRef FV_FindUnresolvedXRef FV_FindAnyTextInset FV_FindUnresolvedTextInset FV_FindAnyPub FV_FindAnyVariable FV_FindAnchoredFrame FV_FindFootnote FV_FindAnyTable FV_FindAutomaticHyphen FV_FindAnyRubi
FS_FindMarkerOfType	FT_String marker type
FS_FindMarkerText	FT_String marker text
FS_FindXRefWithFormat	FT_String format
FS_FindNamedVariable	FT_String variable name

Property	Meaning and possible values
`FS_FindCondTextInCondTags`	`FT_Strings` condition tags
`FS_FindCondTextNotInCond Tags`	`FT_Strings` condition tag

a. Actually including a `FS_FindCharFmt` property is not required; when FrameMaker sees one of the text properties it assumes the client is searching for a character format.
b. See the FDK Programmer's Reference for a description of these properties and their values.

Whenever this function finds something that corresponds to a text range (a word, object anchor, marker, etc.), it returns an `F_TextRangeT` structure for that range. However, when searching for structure elements, you can find elements that have no corresponding text range.

Structure elements for the following table parts have no corresponding text range:

- Table title
- Table head
- Table foot
- Table body
- Table row
- Table cell

When you find a structure element for one of these objects, `F_ApiFind()` returns an empty `F_TextRangeT` structure and `FA_errno` is set to `FE_Success`. In this case, you can get the document's `FP_ElementSelection` property to return a corresponding `F_ElementRangeT` structure for the table part structure element.

Returns

When it finds anything other than structure elements for table parts, an `F_TextRangeT` structure. When it finds structure elements for table parts, an empty `F_TextRangeT` structure, and `FA_errno` is set to `FE_Success`.

If `F_ApiFind()` fails, an empty text range is returned and `FA_errno` is set to one of the following values.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadParameter`	`findParamsp` was empty or a parameter was improperly specified
`FE_BadInsertPos`	The `textLocp` was not valid
`FE_NotTextObject`	The `textLocp` was not a text location

Example

The following code searches the specified document from the beginning of the current selection for the first occurrence of a text string:

```
. . .
F_ObjHandleT docId;
F_PropValsT findParams;
F_TextRangeT trSel, trFound;
StringT searchString = (StringT)"findme";

/* Assuming a document has been attached to a window... */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
if (docId == 0)
   return;

/* Set 'trSel' to the text range of the selection */
trSel = F_ApiGetTextRange(FV_SessionId, docId,
                          FP_TextSelection);
if (trSel.beg.objId == 0 && trSel.end.objId == 0)
   return;

/* Allocate and initialize 'findParams' */
findParams = F_ApiAllocatePropVals(1);
findParams.val[0].propIdent.num = FS_FindText;
findParams.val[0].propVal.valType = FT_String;
findParams.val[0].propVal.u.sval =
                          F_StrCopyString(searchString);

/* Perform the text string search */
trFound = F_ApiFind(docId, &trSel.beg, &findParams);

/* Free the F_PropValsT structures */
F_ApiDeallocatePropVals(&findParams);
. . .
```

F_ApiGetAllKeyDefinitions()

`F_ApiGetAllKeyDefinitions()` gets all the key definitions from the specified key catalog.

Synopsis

```
#include "fapi.h"
. . .
F_TypedValsT F_ApiGetAllKeyDefinitions(F_ObjHandleT
keyCatalogId, IntT filterType)
```

Arguments

`keyCatalogId`	The ID of the key catalog from which to get the key definitons.
`filterType`	Specifies the type of key fields to get for each key definition. `flterType` can have the following values: • `FV_KeyDefFieldsTypePrimary`: Gets only the primary key fields (`Tag`, `Target`, `SrcFile`, and `Duplicate`) • `FV_KeyDefFieldsTypeAll`: Gets all key fields.

Returns

Returns the information in an `F_TypedValsT` structure as follows: `FieldTag` is of type `FT_Integer`. `FieldValue` is of type as specified in the table below. .

FieldTag value	FieldValue type
`FV_KeydefKeyAttrs`	`FT_AttributesEx`
`FV_KeydefKeyDefaultText`	`FT_String`
`FV_KeydefKeyDuplicate`	`FT_Integer`
`FV_KeydefKeyFoundInRefFile`	`FT_Integer`
`FV_KeydefKeyInValid`	`FT_Integer`
`FV_KeydefKeySrcFile`	`FT_String`
`FV_KeydefKeySrcType`	`FT_Integer`
`FV_KeydefKeyTag`	`FT_String`
`FV_KeydefKeyTarget`	`FT_String`
`FV_KeydefKeyVarList`	`FT_Vals`

If `F_ApiGetAllKeyDefinitions()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadObjId`	The ID provided does not specify a key catalog.
`FE_KeyCatalogNotLoaded`	The key catalog provided is currently not loaded.
`FE_KeyCatalogIsStale`	The key catalog provided is currently marked as stale and needs to be re-loaded before using.
`FE_BadFilterType`	The filter type provided is not valid.

Structured F_ApiGetAttributeDefs()

`F_ApiGetAttributeDefs()` gets an element definition's attribute definitions.

Synopsis

```
#include "fapi.h"
. . .
F_AttributeDefsT F_ApiGetAttributeDefs(
   F_ObjHandleT docId,
   F_ObjHandleT elemDefId);
```

Arguments

`docId`	The ID of the document or book containing the element definition
`elemDefId`	The ID of the element definition for which to get attribute definitions

Returns

An `F_AttributeDefsT` structure specifying the attribute definitions. The `F_AttributeDefsT` structure is defined as:

```
typedef struct {
   UIntT len;
   F_AttributeDefT *val;
} F_AttributeDefsT;
```

The `F_AttributeDefT` structure describes an attribute definition. It is defined as:

```
typedef struct {
   StringT name; /* The attribute name. */
   BoolT required; /* True if the attribute is required. */
   BoolT readOnly; /* True if the attribute is read only. */
   IntT attrType; /* The attribute type. See following table. */
   F_StringsT choices;/* Choices if attrType is FV_AT_CHOICES.*/
   F_StringsT defValues;/* The default if the attribute is not
                        ** required. If attrType is
                        ** FV_AT_REALS, FV_AT_STRINGS, etc,
                        ** the default can have multiple strings.
                        */
   StringT rangeMin; /* The minimum allowed value (if any). */
   StringT rangeMax; /* The maximum allowed value (if any). */
} F_AttributeDefT;
```

The `F_AttributeDefT.attrType` field identifies the attribute value's type. It can specify one of the following constants.

attrType constant	Attribute value type
`FV_AT_STRING`	Any arbitrary text string
`FV_AT_STRINGS`	One or more arbitrary text strings
`FV_AT_CHOICES`	A value from a list of choices
`FV_AT_INTEGER`	A signed whole number (optionally restricted to a range of values)
`FV_AT_INTEGERS`	One or more integers (optionally restricted to a range of values)
`FV_AT_REAL`	A real number (optionally restricted to a range of values)
`FV_AT_REALS`	One or more real numbers (optionally restricted to a range of values)
`FV_AT_UNIQUE_ID`	A string that uniquely identifies the element
`FV_AT_UNIQUE_IDREF`	A reference to a UniqueID attribute
`FV_AT_UNIQUE_IDREFS`	One or more references to UniqueID attributes

If `F_ApiGetAttributeDefs()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_WrongProduct`	The current FrameMaker product doesn't support the requested operation.
`FE_BadObjId`	Invalid object ID

Example

The following code gets the attribute definitions from the Chapter element definition and prints the names of the required attributes:

```
. . .
F_ObjHandleT docId, edefId;
F_AttributeDefsT attributeDefs;
UIntT i;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
edefId = F_ApiGetNamedObject(docId, FO_ElementDef, "Chapter");
attributeDefs = F_ApiGetAttributeDefs(docId, edefId);

for(i=0; i<attributeDefs.len; i++)
{
    if(attributeDefs.val[i].required)
        F_Printf(NULL, "%s is a required attribute.\n",
               attributeDefs.val[i].name);
}
. . .
```

See also

F_ApiSetAttributeDefs().

Structured F_ApiGetAttributes()

`F_ApiGetAttributes()` gets an element's attributes.

Synopsis

```
#include "fapi.h"
. . .
F_AttributesT F_ApiGetAttributes(
    F_ObjHandleT docId,
    F_ObjHandleT elemId);
```

Arguments

`docId`	The ID of the document or book containing the element
`elemId`	The ID of the element for which to get attributes

Returns

An `F_AttributesT` structure specifying the attributes. The `F_AttributesT` structure is defined as:

```
typedef struct {
    UIntT len;
    F_AttributeT *val;
} F_AttributesT;
```

The `F_AttributeT` structure describes an attribute. It is defined as:

```
typedef struct {
    StringT name; /* The attribute name. */
    F_StringsT values; /* The attribute values. */
    UByteT valflags; /* typedef structValidation error flags
(read-only). */
    UByteT allow; /* Allow error as special case to suppress
                  ** reporting by validation */
} F_AttributeT;
```

The `F_AttributesT` structure returned by `F_ApiGetAttributeDefs()` includes the attributes (`F_AttributeT` structures) in the following order:

- Attributes defined in the element definition in the same order in which they are defined in the element definition
- Undefined attributes in random order

If an element does not have attributes, the `len` field of the `F_AttributesT` structure is set to `0` and the `val` field is set to `NULL`.

You can query the valflags field of an F_AttributeT structure to determine whether the attribute is valid, or what validation errors it might have. The validation error flags are as follows:

valflags flag	Meaning
FV_AV_REQUIRED	This attribute is required, but it has no value assigned to it
FV_INVALID_CHOICE	At least one value for the attribute is not one of the allowed choices
FV_INVALID_FORMAT	The attribute value is the wrong type for the attribute
FV_AV_IDREF_UNRESOLVED	The attribute refers to an undefined ID value
FV_AV_ID_DUPLICATE_IN_DOC	The attribute value should be unique, but is not unique within the document
FV_AV_TOO_MANY_TOKENS	The attribute value has more than one token, but the attribute definition only allows one token
FV_AV_UNDEFINED	The attribute is not defined for the containing element
FV_AV_OUT_OF_RANGE	The attribute value is out of the range specified in the attribute definition

If `F_ApiGetAttributes()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
FE_BadDocId	Invalid document ID
FE_WrongProduct	The current FrameMaker product doesn't support the requested operation.
FE_BadObjId	Invalid object ID

Example

The following code gets the attributes from the selected element and prints their names:

```
. . .
F_ObjHandleT docId;
F_AttributesT attributes;
F_ElementRangeT er;
UIntT i;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
er = F_ApiGetElementRange(FV_SessionId,
          docId, FP_ElementSelection);
attributes = F_ApiGetAttributes(docId, er.beg.childId);

for(i=0; i<attributes.len; i++)
{
       F_Printf(NULL, "Attribute %d: %s.\n", i,
              attributes.val[i].name);
}
. . .
```

See also

F_ApiSetAttributes().

Structured F_ApiGetElementCatalog()

`F_ApiGetElementCatalog()` gets the list of current valid elements at the insertion point by querying a document's `FP_ElementCatalog` property.

The list matches the list displayed in the Element Catalog on the user's screen. Its contents depend on the value of the current document's `FP_ElementCatalogDisplay` property. For example, if the document's `FP_ElementCatalogDisplay` property is set to `FV_ELCAT_CHILDREN`, the list returned by `F_ApiGetElementCatalog()` contains children allowed anywhere in the current parent element.

Synopsis

```
#include "fapi.h"
. . .
F_ElementCatalogEntriesT F_ApiGetElementCatalog(
   F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document whose Element Catalog you want to query

Returns

An `F_ElementCatalogEntriesT` structure.

The `F_ElementCatalogEntriesT` structure is defined as:

```
typedef struct {
   UIntT len;
   F_ElementCatalogEntryT *val;
} F_ElementCatalogEntriesT;
```

The `F_ElementCatalogEntryT` structure describes an individual catalog entry in a Structured FrameMaker element catalog. It is defined as:

```
typedef struct {
   F_ObjHandleT objId; /* ID of element definition. */
   IntT flags; /* Validation type. */
} F_ElementCatalogEntryT;
```

The `F_ElementCatalogEntryT.flags` field can specify one of the following constants.

flags constant	Meaning
`FV_STRICTLY_VALID`	Catalog entry is strictly valid
`FV_LOOSELY_VALID`	Catalog entry is loosely valid
`FV_ALTERNATIVE`	Catalog entry is an alternative
`FV_INCLUSION`	Catalog entry is valid because it is an inclusion

If no flags are set, the element is invalid at the current position.

If `F_ApiGetElementCatalog()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_WrongProduct`	The current FrameMaker product doesn't support the requested operation.

Example

The following code gets the list of current valid elements at the insertion point in the active document:

```
. . .
F_ObjHandleT docId;
F_ElementCatalogEntriesT validEntries;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
validEntries = F_ApiGetElementCatalog(docId);
. . .
```

Structured F_ApiGetElementRange()

`F_ApiGetElementRange()` gets an element range (`F_ElementRangeT`) property.

Synopsis

```
#include "fapi.h"
. . .
F_ElementRangeT F_ApiGetElementRange(
    F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query. If the object is a session, specify `FV_SessionId`.
`objId`	The ID of the object whose property you want to query.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_ElementSelection`.

Returns

An `F_ElementRangeT` structure specifying the element range. The `F_ElementRangeT` structure is defined as:

```
typedef struct {
    F_ElementLocT beg; /* Beginning of the element range. */
    F_ElementLocT end; /* End of the element range. */
} F_ElementRangeT;
```

The `F_ElementLocT` structure specifies a location within an element. It is defined as:

```
typedef struct {
    F_ObjHandleT parentId; /* Parent element ID. */
    F_ObjHandleT childId; /* Child element ID. */
    IntT offset; /* Offset within child/parent element. */
} F_ElementLocT;
```

If the selection includes the root element, `beg.parentId` is `0`, `beg.childId` is the ID of the root element, and `end.childId` is `0`.

If `F_ApiGetElementRange()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_WrongProduct`	The current FrameMaker product doesn't support the requested operation.
`FE_BadObjId`	Invalid object ID

Example

The following code prints the names of the selected elements and the offsets of the selection within the elements:

```
. . .
F_ObjHandleT docId;
F_ElementRangeT er;
StringT edefName;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
er = F_ApiGetElementRange(FV_SessionId,
          docId, FP_ElementSelection);

edefName = F_ApiGetString(docId, F_ApiGetId(docId,
                er.beg.parentId, FP_ElementDef), FP_Name);
F_Printf(NULL, "Beginning parent element tag: %s\n", edefName);
F_Free(edefName);
edefName = F_ApiGetString(docId, F_ApiGetId(docId,
                er.beg.childId, FP_ElementDef), FP_Name);
F_Printf(NULL, "Beginning child element tag: %s\n", edefName);
F_Free(edefName);
F_Printf(NULL, "Beginning offset: %d\n", er.beg.offset);

edefName = F_ApiGetString(docId, F_ApiGetId(docId,
                er.end.parentId, FP_ElementDef), FP_Name);
F_Printf(NULL, "End parent element tag: %s\n", edefName);
F_Free(edefName);
edefName = F_ApiGetString(docId, F_ApiGetId(docId,
                er.end.childId, FP_ElementDef), FP_Name);
F_Printf(NULL, "End child element tag: %s\n", edefName);
F_Free(edefName);
F_Printf(NULL, "End.offset: %d\n", er.end.offset);
. . .
```

See also

F_ApiSetElementRange().

F_ApiGetAllKeys()

Gets all the key tags from the specified key catalog.

Synopsis

```
#include "fencode.h"
. . .
F_StringsT F_ApiGetAllKeys(F_ObjHandleT keyCatalogId)
```

Arguments

`keyCatalogId` The ID of the key catalog from which to get all key tags.

Returns

Returns the list of key tags as `F_StringsT`. The returned list does not contain duplicate tags.

If `F_ApiGetAllKeys()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadObjId`	The ID provided does not specify a key catalog.
`FE_KeyCatalogNotLoaded`	The key catalog provided is currently not loaded.
`FE_KeyCatalogIsStale`	The key catalog provided is currently marked as stale and needs to be re-loaded before using.

F_ApiGetEncodingForFamily()

`F_ApiGetEncodingForFamily()` returns the encoding the FrameMaker product uses for the font family.

Note that to get the list of font families, angles, variations, or weights in the current FrameMaker product session, you use `F_ApiGetStrings()`. For more information, see F_ApiGetStrings().

Synopsis

```
#include "fencode.h"
. . .
StringT F_ApiGetEncodingForFamily(IntT family);
```

Arguments

`family`	The font family for which you want to know the encoding

Returns

The following strings which indicate encoding for text fonts:

Value	Means
`FrameRoman`	Roman text
`JISX0208.ShiftJIS`	Japanese text
`BIG5`	Traditional Chinese text
`GB2312-80.EUC`	Simplified Chinese text
`KSC5601-1992`	Korean text
`Multiple`	More than one encoding for the font family

If the returned string is Multiple, the font family includes variations that are represented by different encodings. In that case, you should use `F_ApiFamilyFonts()` to get a list of the variations for the family. Then you can use `F_ApiGetEncodingForFont()` to get the encoding for a specific variation.

Non-text fonts may return `FrameRoman`, or they may return the font family name. For example, on some platforms the encoding for the Symbol font family is indicated by the string `Symbol`.

Example

The following code gets the index for the Minchu font family from the session list of font families. It then gets the encoding for that font family:

```
. . .
#include "futils.h"
#include "fstrings.h"
#include "fencode.h"
. . .
F_StringsT families;
StringT encoding;
UIntT i;
/* First get the list of font families for the session */
families = F_ApiGetStrings(0, FV_SessionId, FP_FontFamilyNames);
/* Now get the index of the Minchu family */
for (i=0; i < families.len; i++)
   if (F_StrIEqual(families.val[i], (StringT) "minchu")) break;
if (i == families.len) return; /* Minchu not found */
/* Now use the index to get the encoding for Minchu */
encoding = F_ApiGetEncodingForFamily(i);
/* Free the strings */
F_ApiDeallocateStrings(&families);
F_ApiDeallocateString(&encoding);
```

F_ApiGetEncodingForFont()

`F_ApiGetEncodingForFont()` returns the encoding the FrameMaker product uses for a specific font with a specific combination of weight, angle, and variation.

Note that to get the permutations of weights, angles, and variations for a font, use `F_ApiFamilyFonts()`. Then loop through the list to get an instance with a specific combination of these values. For more information, see F_ApiFamilyFonts().

Synopsis

```
#include "fencode.h"
. . .
StringT F_ApiGetEncodingForFont(F_FontT *font);
```

Arguments

`font`	Pointer to an individual font with a specific combination of weight, angle, and variation

Returns

The following strings which indicate the encoding for text fonts:

Value	Means
`FrameRoman`	Roman text
`JISX0208.ShiftJIS`	Japanese text
`BIG5`	Traditional Chinese text
`GB2312-80.EUC`	Simplified Chinese text
`KSC5601-1992`	Korean text

Non-text fonts may return `Roman`, or they may return the font family name. For example, on some platforms the encoding for a permutation of the Symbol font family is indicated by the string `Symbol`.

Example

The following code gets the index for the Minchu font family from the session list of font families. If the encoding is Multiple, it loops through the permutations of Minchu and prints the encoding for each to the console. If the encoding is not Multiple, it prints out the encoding for Minchu.

```
. . .
#include "futils.h"
#include "fstrings.h"
#include "fencode.h"
. . .
F_FontsT fam;
F_StringsT families, weights, variations, angles;
StringT encoding;
UIntT i, j;

/* Get lists of families, variations, weights, and angles. */
families = F_ApiGetStrings(0, FV_SessionId, FP_FontFamilyNames);
weights = F_ApiGetStrings(0, FV_SessionId, FP_FontWeightNames);
variations = F_ApiGetStrings(0, FV_SessionId,
                        FP_FontVariationNames);
angles = F_ApiGetStrings(0, FV_SessionId,FP_FontAngleNames);

/* Get the index for Minchu */
for (i=0; i < families.len; i++)
   if (F_StrIEqual(families.val[i], (StringT) "minchu")) break;
if (i == families.len) return; /* Minchu not found */

/* Now get the encoding for Minchu... If multiple, print */
/* the encoding for each variation to the console */
encoding = F_ApiGetEncodingForFamily(i);
if (F_StrIEqual(encoding, (StringT) "multiple")) {
   fam = F_ApiFamilyFonts(i);
   for (j = 0; j < fam.len; j++) {
       F_ApiDeallocateString(&encoding);
       encoding = F_ApiGetEncodingForFont(fam.val[j]);
       F_Printf(NULL, "The encoding for %s-%s-%s-%s is %s\n"
```

```
            families.val[fam.val[j].family],
            weights.val[fam.val[j].weight],
            variations.val[fam.val[j].variation],
            angles.val[fam.val[j].angle],
            encoding);
    }
    /* Free the F_FontsT structure. */
    F_ApiDeallocateFonts(&fam);
}
else F_Printf(NULL, "The encoding for Minchu is %s\n" encoding);
/* Free the structures and strings */
F_ApiDeallocateStrings(&families);
F_ApiDeallocateStrings(&weights);
F_ApiDeallocateStrings(&variations);
F_ApiDeallocateStrings(&angles);
F_ApiDeallocateString(&encoding);
```

F_ApiGetId()

`F_ApiGetId()` queries an ID (`F_ObjHandleT`) property.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiGetId(F_ObjHandleT docId,
       F_ObjHandleT objId,
       IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query. If the object is a session, specify `FV_SessionId`.
`objId`	The ID of the object whose property you want to query.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_ActiveDoc`.

Returns

The ID specified by the property. If the property doesn't specify an ID or an error occurs, `F_ApiGetId()` returns `0`.

IMPORTANT: *If* `F_ApiGetId()` *returns* `0`*, it may not indicate an error, because some ID property values can be* `0`*. To determine if a returned* `0` *is a property value or an error, check* `FA_errno`*.*

If `F_ApiGetId()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code displays the color of the active document's change bars:

```
. . .
#include "futils.h"
F_ObjHandleT docId, colorId;
UCharT msg[256];
StringT color;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Get the ID of the document's change bar color. */
colorId = F_ApiGetId(FV_SessionId,
                     docId, FP_ChangeBarColor);

/* Get the color's name and display it. */
color = F_ApiGetString(docId, colorId, FP_Name);
F_Sprintf(msg, "The document's change bar color is %s.",color);
F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);

F_Free(color);
. . .
```

See also

F_ApiGetNamedObject() and F_ApiSetId().

F_ApiGetImportDefaultParams()

`F_ApiGetImportDefaultParams()` gets a default property list that you can use to call `F_ApiImport()`.

Synopsis

```
#include "fapi.h"
. . .
F_PropValsT F_ApiGetImportDefaultParams();
```

Arguments

None.

Returns

A property list (an `F_PropValsT` data structure) with the properties shown in the following tables. The first value listed by each property is the value that `F_ApiGetImportDefaultParams()` assigns to the property. The other values are values you can set the property to.

The property list contains all the properties shown in these tables. However, `F_ApiImport()` uses some of the properties only for certain types of import operations. For example, it uses the `FS_UseMainFlow` property only if you are importing a Frame document or MIF file; it ignores this property if you are importing a text file or a graphic.

`F_ApiImport()` uses the following properties for all import operations.

Property	Instruction or situation and possible values
`FS_AlertUserAboutFailure`	Alert user if an unexpected condition, such as an unrecognized file type, occurs.
	`False`: don't notify user when unexpected conditions occur.
	`True`: notify user when unexpected conditions occur.
`FS_DisallowDoc`	Disallow Frame binary documents.
	`False`: allow them to be imported.
	`True`: don't allow them to be imported.
`FS_DisallowFilterTypes`	Disallow filterable files.
	`False`: allow them to be imported.
	`True`: don't allow them to be imported.
`FS_DisallowMIF`	Disallow MIF files.
	`False`: allow them to be imported.
	`True`: don't allow them to be imported.
`FS_DisallowGraphicTypes`	Disallow graphic files.
	`False`: allow them to be imported.
	`True`: don't allow them to be imported.

Property	Instruction or situation and possible values
FS_DisallowPlainText	Disallow Text Only files.
	False: allow them to be imported.
	True: don't allow them to be imported.
FS_DisallowSgml	Disallow SGML documents.
	False: allow them to be imported.
	True: don't allow them to be imported.
FS_DisallowXML	Disallow XML documents.
	False: allow them to be imported.
	True: don't allow them to be imported.
FS_DontNotifyAPIClients	Notify other clients of the import operation.
	True: don't notify them.
	False: notify them.
FS_FileTypeHint	If the file is filterable, a string that enables the FrameMaker product to automatically call the correct filter to filter it. For information on the syntax of this string, see Syntax of FP_ImportHint strings and Syntax of FP_ImportHint strings.
	NULL.
FS_FileIsSgmlDoc	File is an SGML document.
	FV_DoOK: import it anyway.
	FV_DoCancel: cancel import operation.
	FV_DoShowDialog: show dialog box and let user decide.
FS_FileIsXMLDoc	File is an XML document.
	FV_DoOK: import it anyway.
	FV_DoCancel: cancel import operation.
	FV_DoShowDialog: show dialog box and let user decide.

Property	Instruction or situation and possible values
FS_ForceImportAsText	Import file as a Text Only document, even if it is a MIF file or a filterable file.[a]
	False: import it in a format based on its type
	True: import it as Text Only.
FS_HowToImport	Import file by reference or copy.
	FV_DoByRef: import file by reference.
	FV_DoByCopy: import file by copy.
	FV_DoUserChoice: allow user to choose how to import the file.
FS_ImportAsType	Specify the format of the file to import.
	FV_AUTORECOGNIZE: Default value; recognize the file type aoutmatically.
	FV_TYPE_BINARY: A FrameMaker binary file.
	FV_TYPE_FILTER: Use a filter to import this file. You must specify a valid file type hint for FS_FileTypeHint.
	FV_TYFE_MIF
	FV_TYPE_TEXT
	FV_TYPE_SGML
	FV_TYPE_XML
FS_InsetData	Used as an import parameter in import script. The value of this parameter is used as INSETInfo Default: 0
FS_NumImportParams	The available number of properties in the import script that the user can set
FS_NumImportReturnParams	The available number of properties in the import-return script that the user can set
FS_ManualUpdate	Update inset manually.
	False: don’t update inset manually.
	True: update inset manually.

Property	Instruction or situation and possible values
FS_SgmlImportApplication	
	Retained for compatibility. Use FS_StructuredImportApplication
	.
FS_ShowBrowser	Display Import dialog box.
	False: don't display it.
	True: display it.
FS_TextInsetName	Inset name.
	NULL.

a. Some file types, such as Frame binary files, can't be imported as text.

F_ApiImport() uses the following properties only for importing FrameMaker product documents and MIF files.

Property	Instruction or situation and possible values
FS_FileIsMakerDoc	File is a Frame binary document or a MIF file
	FV_DoOK: import it anyway
	FV_DoCancel: cancel import operation
	FV_DoShowDialog: show dialog box and let user decide
FS_FormatImportedText	Format the imported text
	FV_EnclosingDoc: use formatting in the enclosing document
	FV_PlainText: format the imported text as plain text
	FV_SourceDoc: use formatting from the source document
FS_ImportFlowPageSpace	If FS_UseMainFlow is False, the type of pages to search for the flow specified by FS_ImportFlowTag
	FV_BodyPage: search body pages
	FV_ReferencePage: search reference pages

Property	Instruction or situation and possible values
FS_ImportFlowTag	If FS_UseMainFlow is False, the name of the flow to import
	NULL
FS_RemoveManualPageBreaks	Remove manual page breaks if FS_FormatImportedTest is set to FV_EnclosingDoc
	True: remove manual page breaks
	False: don't remove manual page breaks
FS_RemoveOverrides	Remove format overrides if FS_FormatImportedTest is set to FV_EnclosingDoc
	True: remove format overrides
	False: don't remove format overrides
FS_UseMainFlow	Import text from specified document's main flow
	True: import the text from the main flow
	False: don't import the text from the main flow

F_ApiImport() uses the following properties only for importing graphics files

Property	Instruction or situation and possible values
FS_FileIsGraphic	File is a graphic file
	FV_DoOK: import it
	FV_DoCancel: cancel import operation
	FV_DoShowDialog: display dialog box and let user decide
FS_FitGraphicInSelectedRect	Fit the graphic in the selected graphic frame
	True: fit the graphic in the frame
	False: don't fit the graphic in the frame

Property	Instruction or situation and possible values
FS_GraphicDpi	Dots per inch (DPI) at which to import the graphic
	72
FS_PDFPageNum	The page number to be imported from a PDF file. Used when you want to import a specific page of a PDF file.

F_ApiImport() uses the following properties only for importing ASCII text files.

Property	Instruction or situation and possible values
FS_CellSeparator	If FS_FileIsText is FV_DoImportAsTable, the delimiter or separator used to parse the text into cells
	NULL
FS_FileIsText	File is a Text Only file
	FV_TextFile_EOLisEOP: import it and convert each end-of-line into a paragraph break
	FV_TextFile_EOLisNotEOP: import it but don't convert each end-of-line into a paragraph break
	FV_DoImportAsTable: import it into a table.
	FV_DoCancel: cancel Import operation
FS_ImportTblTag	If FS_FileIsText is FV_DoImportAsTable, the table format to use
	NULL
FS_LeaveHeadingRowsEmpty	If FS_FileIsText is FV_DoImportAsTable, leave heading rows empty
	False: don't leave heading rows empty
	True: leave heading rows empty
FS_NumCellSeparators	If FS_FileIsText is FV_DoImportAsTable, and FS_CellSeparator is set to a space (' '), the number of spaces to use as a separator
	1

Property	Instruction or situation and possible values
FS_NumColumns	If FS_FileIsText is FV_DoImportAsTable, and FS_TreatParaAsRow is False, the number of columns in the table
	1
FS_TblNumHeadingRows	If FS_FileIsText is FV_DoImportAsTable, the number of heading rows in the table
	1
FS_TreatParaAsRow	If FS_FileIsText is FV_DoImportAsTable, convert each line in the text file into a row of table cells and use FS_CellSeparator and FS_NumCellSeparators to determine how to divide the line into separate cells
	True: convert each line into a row of table cells
	False: convert each line into a table cell instead

When you import text into a table, in addition to setting FS_FileIsText to FV_DoImportAsTable, you must specify a value for the FS_ImportTblTag property. If you set the property FS_TreatParaAsRow to True, you must also specify a value for the FS_CellSeparator property.

The property list returned by F_ApiGetImportDefaultParams() does not specify values for the FS_ImportTblTag and FS_CellSeparator properties. If you use the property list to import a table and do not specify a value for FS_ImportTblTag, F_ApiImport() fails and sets FA_errno to FE_BadParameter. If you set FS_TreatParaAsRow to True and do not specify a cell separator by setting FS_CellSeparator, F_ApiImport() fails and sets FA_errno to FE_BadParameter.

If F_ApiGetImportDefaultParams() fails, the API sets the len field of the returned F_PropValsT data structure to 0.

After you are done with the property list returned by F_ApiGetImportDefaultParams(), use F_ApiDeallocatePropVals() to deallocate it.

Example

The following code gets a default Import script with `F_ApiGetImportDefaultParams()`. It modifies the script to disallow Frame binary, graphic, and MIF files. It then uses the script to call `F_ApiImport()`.

```
. . .
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT docId;
F_TextRangeT tr;
IntT i;

/* Get default import list. Return if it can't be allocated. */
params = F_ApiGetImportDefaultParams();

/* Get current insertion point. Return if there isn't one. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(0, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Change properties to disallow Frame documents. */
i = F_ApiGetPropIndex(&params, FS_DisallowMIF);
params.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&params, FS_DisallowDoc);
params.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&params, FS_DisallowGraphicTypes);
params.val[i].propVal.u.ival = True;

F_ApiImport(docId, &tr.beg, "/tmp/mydata.txt",
             &params, &returnParamsp);

/* Deallocate property lists. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
. . .
```

See also

F_ApiImport() and F_ApiDeallocateStructureType().

F_ApiGetInt()

`F_ApiGetInt()` queries an integer (`IntT`) property.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiGetInt(F_ObjHandleT docId,
       F_ObjHandleT objId,
       IntT propNum);
```

Arguments

`docId`	The ID of the document, dialog box, book, or session containing the object whose property you want to query. If the object is a session, specify `0`.
`objId`	The ID of the object whose property you want to query.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_FnNum`.

Returns

The value for the specified property, or `0` if an error occurs.

IMPORTANT: *If* `F_ApiGetInt()` *returns* `0`*, it probably does not indicate an error, because most* `IntT` *property values can be* `0`*. To determine if a returned* `0` *is a property value or an error, check* `FA_errno`*.*

If `F_ApiGetInt()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code displays the major version number of the FrameMaker product running in the current session:

```
. . .
#include "futils.h"
UCharT msg[256];
F_Sprintf(msg, "FrameMaker product version %d.",
       F_ApiGetInt(0, FV_SessionId, FP_VersionMajor));
F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
. . .
```

See also

F_ApiSetInt().

F_ApiGetIntByName()

`F_ApiGetIntByName()` queries an integer (`IntT`) facet.

`F_ApiGetIntByName()` and other `F_ApiGet`*`PropertyType`*`ByName()` functions use a transaction model to query facets. After you have finished a series of queries, you must commit the transaction by calling `F_ApiGetIntByName()` to query a facet named `""`.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiGetIntByName(F_ObjHandleT docId,
       F_ObjHandleT objId,
       StringT propName);
```

Arguments

`docId`	The ID of the document containing the inset whose facet you want to query
`objId`	The ID of the inset whose facet you want to query
`propName`	The name of the facet to query

Returns

The value for the specified facet, or `0` if an error occurs.

IMPORTANT: *If* `F_ApiGetIntByName()` *returns* `0`*, it may not indicate an error, because some facet values can be* `0`*. To determine if a returned* `0` *is a property value or an error, check* `FA_errno`*.*

If `F_ApiGetIntByName()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property name is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code queries an integer facet named `revision.facet`:

```
. . .
IntT revision;
F_ObjHandleT docId, insetId;
revision = F_ApiGetIntByName(docId, insetId, "revision.facet");
F_ApiGetIntByName(docId, insetId, ""); /* Commit transaction */
. . .
```

See also

F_ApiSetIntByName().

F_ApiGetInts()

`F_ApiGetInts()` queries an `F_IntsT` (array of integers or object IDs) property.

Synopsis

```
#include "fapi.h"
. . .
F_IntsT F_ApiGetInts(F_ObjHandleT docId,
       F_ObjHandleT objId,
       IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query.
`objId`	The ID of the object whose property you want to query.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_GeneralRuleErrorOffsets`.

Returns

The `F_IntsT` structure for the specified property.

The returned `F_IntsT` structure references memory that is allocated by the API. Use `F_ApiDeallocateInts()` to free this memory when you are done with it.

If `F_ApiGetInts()` fails, the API sets the `len` field of the returned structure to `0` and assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn’t support the operation

Example

The following code displays the condition formats that apply to the text at the insertion point:

```
. . .
F_ObjHandleT docId;
UCharT msg[256];
F_IntsT condIds;
IntT i;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Get IDs of conditions that apply at insertion point. */
condIds = F_ApiGetInts(FV_SessionId, docId, FP_InCond);

/* Get name of each condition and display it. */
for (i=0; i<condIds.len; i++)
   {
   F_Sprintf(msg,"%s condition applies to text you are typing.",
      F_ApiGetString(docId,condIds.val[i], FP_Name));
   F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
   }


F_ApiDeallocateInts(&condIds);
. . .
```

See also

F_ApiSetInts().

F_ApiGetKeyCatalog()

Finds a key catalog with the specified tag.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiGetKeyCatalog(StringT tag)
```

Arguments

`tag`	The tag of the key catalog.

Returns

Returns the handle of the new key catalog.

If `F_ApiGetKeyCatalog()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_BadName`	The tag provided is not valid or the key catalog with this tag does not exist.

F_ApiGetKeyDefinition()

Gets the specified key definition field for the specified key from the specified key catalog.

Synopsis

```
#include "fapi.h"
. . .
F_TypedValT F_ApiGetKeyDefinition(F_ObjHandleT keyCatalogId,
StringT key, IntT keyField)
```

Arguments

`keyCatalogId`	The ID of the key catalog from which to get the key definiton.
`key`	The tag of the key for which the key definition is being asked for.
`keyField`	The key field (or key information) that is being asked for.

The valid keyField values and the corresponding value type are as follows:

keyField	Value type
`FV_KeydefKeyTag`	`FT_String`
`FV_KeydefKeyTarget`	`FT_String`
`FV_KeydefKeySrcFile`	`FT_String`
`FV_KeydefKeyDuplicate`	`FT_Integer`

keyField	Value type
FV_KeydefKeySrcType	FT_Integer
FV_KeydefKeyVarList	FT_Vals
FV_KeydefKeyDefaultText	FT_String
FV_KeydefKeyFoundInRefFile	FT_Integer
FV_KeydefKeyInValid	FT_Integer
FV_KeydefKeyAttrs	FT_AttributesEx

Returns

Returns the requested value in a F_TypedVal structure as per the value type specified in the previous table.

If F_ApiGetKeyDefinition() fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
FE_BadObjId	The ID provided does not specify a key catalog.
FE_BadKey	The key provided is not valid.
FE_KeyCatalogNotLoaded	The key catalog provided is currently not loaded.
FE_KeyCatalogIsStale	The key catalog provided is currently marked as stale and needs to be re-loaded before using.
FE_KeyDefinitionDoesNotExist	The definition for the specified key is not available in the key catalog.
FE_WrongProduct	(only for keyField="FV_KeydefKeyAttrs") Current FrameMaker product doesn't support the operation.
FE_BadKeyField	The key field provided is not valid.

F_ApiGetMetric()

`F_ApiGetMetric()` queries a metric (`MetricT`) property.

Synopsis

```
#include "fapi.h"
. . .
MetricT F_ApiGetMetric(F_ObjHandleT docId,
       F_ObjHandleT objId,
       IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query.
`objId`	The ID of the object whose property you want to query.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_Leading`.

Returns

The value of the specified property, or `0` if an error occurs.

IMPORTANT: *If* `F_ApiGetMetric()` *returns* `0`*, it may not indicate an error, because some property values can be* `0`*. To determine if a returned* `0` *is a property value or an error, check* `FA_errno`*.*

If `F_ApiGetMetric()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code displays the indent of the Body paragraph format:

```
. . .
#include "futils.h"
#define in (RealT) (65536*72) /* Frame metric inch */
UCharT msg[256];
F_ObjHandleT docId, fmtId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
fmtId = F_ApiGetNamedObject(docId, FO_PgfFmt, "Body");
F_Sprintf(msg, "Indent: %2.2f inches",
       F_ApiGetMetric(docId, fmtId, FP_LeftIndent)/in);
F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
. . .
```

See also

F_ApiSetMetric().

F_ApiGetMetricByName()

`F_ApiGetMetricByName()` queries a metric (`MetricT`) facet.

`F_ApiGetMetricByName()` and other `F_ApiGet`*`PropertyType`*`ByName()` functions use a transaction model to query facets. After you have finished a series of queries, you must commit the transaction by using `F_ApiGetIntByName()` to query a facet named `""`.

Synopsis

```
#include "fapi.h"
. . .
MetricT F_ApiGetMetricByName(F_ObjHandleT docId,
       F_ObjHandleT objId,
       StringT propName);
```

Arguments

`docId`	The ID of the document containing the inset whose facet you want to query
`objId`	The ID of the inset whose facet you want to query
`propName`	The name of the facet to query

Returns

The value for the specified facet, or `0` if an error occurs.

IMPORTANT: *If* `F_ApiGetMetricByName()` *returns* `0`*, it may not indicate an error, because some facet values can be* `0`*. To determine if a returned* `0` *is a property value or an error, check* `FA_errno`*.*

If `F_ApiGetMetricByName()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property name is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code queries a metric facet named `height.facet`:

```
. . .
MetricT height;
F_ObjHandleT docId, insetId;
height = F_ApiGetMetricByName(docId, insetId, "height.facet");
F_ApiGetIntByName(docId, insetId, ""); /* Commit */
. . .
```

See also

F_ApiSetMetricByName().

F_ApiGetMetrics()

`F_ApiGetMetrics()` queries a metrics (`F_MetricsT`) property.

Synopsis

```
#include "fapi.h"
. . .
F_MetricsT F_ApiGetMetrics(F_ObjHandleT docId,
       F_ObjHandleT objId,
       IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query
`objId`	The ID of the object whose property you want to query
`propNum`	The property to query (for example, `FP_TblColWidths`)

Returns

An `F_MetricsT` data structure, containing the `MetricT` values.

The returned `F_MetricsT` structure references memory that is allocated by the API. Use `F_ApiDeallocateMetrics()` to free this memory when you are done with it.

If `F_ApiGetMetrics()` fails, the API sets the `len` field of the returned structure to `0` and assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code displays information about the dash pattern of the selected graphic object:

```
. . .
#include "futils.h"
#define pts (RealT)(65536)
F_ObjHandleT docId, objId;
UCharT msg[256];
F_MetricsT metrics;
IntT i;

/* Get IDs of active document and selected object. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
objId = F_ApiGetId(FV_SessionId, docId,
                   FP_FirstSelectedGraphicInDoc);

if (!objId) return; /* Return if no object is selected */

/* Get selected object's dash pattern and display it. */
metrics = F_ApiGetMetrics(docId, objId, FP_Dash);

for (i=0; i<metrics.len; i++)
{
   F_Sprintf(msg,"%dth dash or space in pattern is %2.2f pts",
             i, metrics.val[i]/pts);
   F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
}
F_ApiDeallocateMetrics(&metrics);
. . .
```

See also

F_ApiSetMetrics().

F_ApiGetNamedObject()

`F_ApiGetNamedObject()` gets the ID of an object with a specified name (`FP_Name` property) and object type. It works with the following object types:

- `FO_Book`
- `FO_CharFmt`
- `FO_Color`
- `FO_CombinedFontDefn`
- `FO_Command`
- `FO_CondFmt`
- `FO_ElementDef`
- `FO_FmtChangeList`
- `FO_Menu`
- `FO_MenuItemSeparator`
- `FO_MasterPage`
- `FO_PgfFmt`
- `FO_RefPage`
- `FO_RulingFmt`
- `FO_TblFmt`
- `FO_UnanchoredFrame` (reference frame)
- `FO_VarFmt`
- `FO_XRefFmt`

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiGetNamedObject(F_ObjHandleT docId,
    IntT objType,
    StringT name);
```

Arguments

`docId`	The ID of the document containing the object
`objType`	The type of object (for example, `FO_TblFmt`)
`name`	The name of the object whose ID you want to get

Returns

The ID of the object, or `0` if an error occurs.

If `F_ApiGetNamedObject()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_TypeUnNamed`	Objects of the specified type do not have names
`FE_NameNotFound`	Object with the specified name and type doesn't exist in the specified document

Example

The following code gets the ID for a paragraph format named Header and deletes it:

```
. . .
F_ObjHandleT docId, fmtId;
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

fmtId = F_ApiGetNamedObject(docId, FO_PgfFmt, "Header");
F_ApiDelete(docId, fmtId);
. . .
```

See also

F_ApiGetId() and F_ApiGetUniqueObject().

F_ApiGetNewXMLDefaultParams()

`F_ApiGetNewXMLDefaultParams()` generates default open-parameters for `F_ApiNewXML()`.

Synopsis

```
#include "fapi.h"
. . .
F_PropValsT F_ApiGetNewXMLDefaultParams();
```

Returns

A property list (an `F_PropValsT` data structure) with the properties shown in the following table.

Property	Instruction or situation and possible values
`FS_StructuredApplication`	Specifies a structured application to be used for creatinga new XML document
`FS_Doctype`	Specifies a doctype to be used for creating a new XML document
`FS_PublicId`	Specifies a public id to be used for creating a new XML document
`FS_SystemId`	Specifies a DTD-system id to be used for creating a new XML document
`FS_Extension`	Used to provide a custom extension for the new XML document (like `Untitled1.dita`). Without customization, FrameMaker determines the extension by itself based on file type.
`FS_Visible`	A boolean property that indicates if the new XML document shall be visible or hidden.

Example

The following code demonstrates how to create a new XML file with a specified structured application.

```
F_PropValsT params = F_ApiGetNewXMLDefaultParams ();
F_PropValsT *retParams = NULL;
for (UIntT i = 0; i < params.len; i++)
{
   switch (params.val[i].propIdent.num)
   {
      case FS_StructuredApplication:
         params.val[i].propVal.u.sval = F_StrCopyString
("My_Strcutured_App");
         break;
   }
}
F_ApiNewXML (&params, &retParams);
F_ApiDeallocatePropVals (&params);

F_ApiDeallocatePropVals (retParams);
```

F_ApiGetObjectType()

`F_ApiGetObjectType()` returns an object's type.

Synopsis

```
#include "fapi.h"
. . .
UIntT F_ApiGetObjectType(F_ObjHandleT docId,
   F_ObjHandleT objectId);
```

Arguments

`docId`	The ID of the session, document, or dialog box containing the object
`objectId`	The object whose type you want to get

Returns

The object type, such as `FO_Rectangle` or `FO_Pgf`. For a complete list of object types, see Chapter 3, "Object Reference."

Example

The following code notifies the user if the first selected object is a text column:

```
. . .
F_ObjHandleT docId, objId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
objId = F_ApiGetId(FV_SessionId, docId,
                   FP_FirstSelectedGraphicInDoc);
if (F_ApiGetObjectType(docId, objId) == FO_TextFrame)
   F_ApiAlert("Object is text column.", FF_ALERT_CONTINUE_WARN);
. . .
```

F_ApiGetOpenDefaultParams()

`F_ApiGetOpenDefaultParams()` gets a default property list that you can use to call `F_ApiOpen()`.

Synopsis

```
#include "fapi.h"
. . .
F_PropValsT F_ApiGetOpenDefaultParams();
```

Arguments

None.

Returns

A property list (an `F_PropValsT` data structure) with the properties shown in the following table. The first value listed by each property is the value that `F_ApiGetOpenDefaultParams()` assigns to the property. The other values are values you can set the property to.

F_ApiGetOpenDefaultParams() property	Instruction or situation and possible values
`FS_AlertUserAboutFailure`	Alert user if unexpected condition, such as an unrecognized file type, occurs.
	`False`: don't notify user when unexpected conditions occur.
	`True`: notify user when unexpected conditions occur.

F_ApiGetOpenDefaultParams() property	Instruction or situation and possible values
FS_BookIsInUse	Book is already open.
	FV_OpenViewOnly: open a view-only copy of the book.
	FV_OpenEditableCopy: open an editable copy of the book.
	FV_DoShowDialog: show dialog box and let user decide.
	FV_ResetLockAndContinue: reset lock and open file.
	FV_DoCancel: cancel Open operation.
FS_DisallowBookDoc	File is a Frame binary book.
	False: open it anyway.
	True: don't open it.
FS_DisallowBookMIF	File is a MIF book.
	False: open it anyway.
	True: don't open it.
FS_DisallowDoc	File is a Frame binary document.
	False: open it anyway.
	True: don't open it.
FS_DisallowFilterTypes	File is filterable.
	False: open it anyway.
	True: don't open it.
FS_DisallowMIF	File is a MIF document.
	False: open it and convert it.
	True: don't open it.
FS_DisallowPlainText	File is Text Only.
	False: open it and convert it.
	True: don't open it.

F_ApiGetOpenDefaultParams() property	Instruction or situation and possible values
FS_DisallowSgml	File is an SGML document.
	False: open it and convert it.
	True: don't open it.
FS_DontNotifyAPIClients	Open file without notifying other clients.
	False: notify other clients.
	True: don't notify other clients.
FS_FileIsInUse	File is already open.
	FV_OpenViewOnly: open in View Only format.
	FV_OpenEditableCopy: open an editable copy.
	FV_DoShowDialog: show dialog box and let user decide.
	FV_ResetLockAndContinue: reset the lock and open the file.
	FV_DoCancel: cancel Open operation.
FS_FileIsOldVersion	File is from previous version of the FrameMaker product.
	FV_DoCancel: cancel Open operation.
	FV_DoOK: open it anyway.
	FV_DoShowDialog: show dialog box and let user decide.
FS_FileIsStructured	File has FrameMaker features, but current product interface isn't Structured FrameMaker.
	FV_OpenViewOnly: open a View Only copy of file.
	FV_DoCancel: cancel Open operation.
	FV_StripStructureAndOpen: remove structure features and open file.
	FV_DoShowDialog: show dialog box and let user decide.

F_ApiGetOpenDefaultParams() property	Instruction or situation and possible values
FS_FileIsText	File is Text Only.
	FV_TextFile_EOLisEOP: open it and convert each end-of-line into a paragraph break.
	FV_TextFile_EOLisNotEOP: open it but don't convert each end-of-line into a paragraph break.
	FV_DoShowDialog: show dialog box and let user decide.
	FV_DoCancel: cancel Open operation.
FS_FileTypeHint	If the file is filterable, a string that enables the FrameMaker product to automatically call the correct filter to filter it. For information on the syntax of this string, see Syntax of FP_ImportHint strings and Syntax of FP_ImportHint strings.
	NULL.
FS_FontChangedMetric	A font metric needs to be changed.
	FV_DoCancel: cancel Open operation.
	FV_DoOK: open the document anyway.
	FV_DoShowDialog: show dialog box and let user decide.
FS_FontNotFoundInCatalog	Catalog contains fonts that aren't available.
	FV_DoCancel: cancel Open operation.
	FV_DoOK: open it anyway.
	FV_DoShowDialog: show dialog box and let user decide.
FS_FontNotFoundInDoc	Document uses unavailable fonts.
	FV_DoCancel: cancel Open operation.
	FV_DoOK: open it anyway.
	FV_DoShowDialog: show dialog box and let user decide.
FS_ForceOpenAsText	Open file as a Text Only document, even if it is MIF file or filterable file.[a]
	False: open it in a format based on its type.
	True: open it as Text Only (use the method specified by FS_FileIsText).

F_ApiGetOpenDefaultParams() property	Instruction or situation and possible values
FS_LanguageNotAvailable	The file uses an unavailable language.
	FV_DoCancel: cancel Open operation.
	FV_DoOK: open it anyway.
	FV_DoShowDialog: show dialog box and let user decide.
FS_LockCantBeReset	Attempted to reset FrameMaker product file lock but wasn't able to.
	FV_DoCancel: cancel Open operation.
	FV_DoShowDialog: show dialog box and let user decide.
	FV_DoOK: open without resetting the lock.
FS_MakeIconic	Make document an icon as soon as it's opened.
	False: open file in an open window.
	True: iconify it.
FS_MakeVisible	Make document or book visible as soon as it's opened.[b]
	True: make visible.
	False: don't make visible.
FS_NameStripe	String specifying the name that appears on the document title bar.
	NULL.
FS_NewDoc	Create new document.
	False: open existing document.A
	True: create a new document.
FS_NumOpenParams	The available number of properties in the open script that user can set.
FS_NumOpenReturnParams	The maximum number of properties that an open-return script can have.

F_ApiGetOpenDefaultParams() property	Instruction or situation and possible values
FS_OpenAsType	Specify the format of the file to import.
	FV_AUTORECOGNIZE: Default value; recognize the file type aoutmatically.
	FV_TYPE_BINARY: A FrameMaker binary file.
	FV_TYPE_FILTER: Use a filter to import this file. You must specify a valid file type hint for FS_FileTypeHint.
	FV_TYFE_MIF
	FV_TYPE_TEXT
	FV_TYPE_SGML
	FV_TYPE_XML
FS_OpenBookViewOnly	Open book in View Only format.
	False: don't open in View Only format.
	True: open in View Only format.
FS_OpenDocViewOnly	Open document in View Only format.
	False: don't open in View Only format.
	True: open in View Only format.
FS_OpenId	The ID of the document in the current window if FS_OpenInNewWindow is set to False.
	0
FS_OpenFileNotWritable	How to handle the case when opening a file the client cannot write to.
	FV_DoCancel: Cancel the open operation
	FV_DoOK: Open the file in read-only format
	FV_DoShowDialog: Display an alert to notify that the file is not writable.
FS_OpenInNewWindow	Open file in new window.
	True: open file in new window.
	False: open file in the window occupied by the document specified by the FS_OpenId property.

F_ApiGetOpenDefaultParams() property	Instruction or situation and possible values
FS_RefFileNotFound	Document imports another file that isn't available.
	FV_DoCancel: cancel Open operation.
	FV_AllowAllRefFilesUnFindable: open anyway and ignore the referenced file.
	FV_DoShowDialog: show dialog box and let user decide.
FS_SgmlOpenApplication	Retained for compatibility. Use FS_StructureOpenApplication
	NULL.(No Application Used)
FS_ShowBrowser	Display Open dialog box.
	False: don't display it.
	True: display it.
FS_TemplateShouldInsertRoot	When opening a template document, this can be set to true (it is false by default). This results in automatic insertion of the root element.
FS_UpdateBrowserDirectory	Update directory displayed in browser dialog box.
	False: don't update it.
	True: update it.
FS_UpdateTextReferences	Update text insets.
	FV_DoUserPreference: update text insets if the document property, FP_DontUpdateTextInsets, is False.
	FV_DoYes: update text insets.
	FV_DoNo: don't update text insets.
FS_UpdateXRefs	Update cross-references.
	FV_DoUserPreference: update cross-references if the document property, FP_DontUpdateXRefs, is False.
	FV_DoYes: update cross-references.
	FV_DoNo: don't update cross-references.

F_ApiGetOpenDefaultParams() property	Instruction or situation and possible values
FS_UseAutoSaveFile	Use Autosave file if it is present.
	FV_DoCancel: cancel the Open operation.
	FV_DoYes: use it.
	FV_DoNo: don't use it.
	FV_DoShowDialog: show dialog box and let user decide.
FS_UseRecoverFile	Use Recover file if it is present.
	FV_DoCancel: cancel Open operation.
	FV_DoYes: use it.
	FV_DoNo: don't use it.
	FV_DoShowDialog: show dialog box and let user decide.

a. Certain file types, such as Frame binary files, can't be opened as text.
b. Invisible documents and books can never be active.

The returned `F_PropValsT` structure references memory that is allocated by the API. Use `F_ApiDeallocatePropVals()` to free this memory when you are done with it.

If `F_ApiGetOpenDefaultParams()` fails, the API sets the `len` field of the returned structure to `0`.

Example

The following code gets a default Open script with `F_ApiGetOpenDefaultParams()`. It modifies the script to instruct the FrameMaker product to prompt the user if unavailable fonts are used in the document. It then uses the script to call `F_ApiOpen()`.

```
. . .
IntT i;
F_PropValsT script, *returnp = NULL;
F_ObjHandleT docId;

/* Get default property list. */
script = F_ApiGetOpenDefaultParams();

/* Get indexes of properties and change them. */
i = F_ApiGetPropIndex(&script, FS_FontNotFoundInDoc);
script.val[i].propVal.u.ival = FV_DoShowDialog;
i = F_ApiGetPropIndex(&script, FS_FontNotFoundInCatalog);
script.val[i].propVal.u.ival = FV_DoShowDialog;

/* Open /tmp/my.doc. */
docId = F_ApiOpen("/tmp/my.doc", &script, &returnp);

/* Free memory used by Open scripts. */
F_ApiDeallocatePropVals(&script);
F_ApiDeallocatePropVals(returnp);
. . .
```

See also

F_ApiOpen().

F_ApiGetPoints()

`F_ApiGetPoints()` queries a points (`F_PointsT`) property. You can use it to get the array of points (or vertices) for a polygon, line, or polyline object.

Synopsis

```
#include "fapi.h"
. . .
F_PointsT F_ApiGetPoints(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query
`objId`	The ID of the object whose property you want to query
`propNum`	The `F_PointsT` property to query (for example, `FP_Points`)

Returns

An `F_PointsT` structure that contains a pointer to an array of the points' coordinate pairs.

The returned `F_PointsT` structure references memory that is allocated by the API. Use `F_ApiDeallocatePoints()` to free this memory when you are done with it.

If `F_ApiGetPoints()` fails, the API sets the `len` field of the returned structure to `0` and assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code displays the coordinates of the selected polygon's vertices:

```
. . .
#include "futils.h"
#define in (RealT)(72*65536)
F_ObjHandleT docId, objId;
UCharT msg[256];
F_PointsT points;
IntT i;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
objId = F_ApiGetId(FV_SessionId, docId,
                   FP_FirstSelectedGraphicInDoc);

/* Make sure object is selected and it is a polygon. */
if (F_ApiGetObjectType(docId, objId) != FO_Polygon)
   {
      F_ApiAlert("Select a polygon.", FF_ALERT_CONTINUE_NOTE);
      return;
   }
/* Get the polygon's array of vertices.*/
points = F_ApiGetPoints(docId, objId, FP_Points);

/* Iterate through vertices, displaying their coordinates. */
for (i=0; i<points.len; i++)
   {
   F_Sprintf(msg, "Coordinates of point %d are %2.2fin,%2.2fin",
             i,
             points.val[i].x/in,
             points.val[i].y/in);
      F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
   }
F_ApiDeallocatePoints(&points);
. . .
```

See also

F_ApiSetPoints().

F_ApiGetPropIndex()

`F_ApiGetPropIndex()` gets the index of a property-value pair (`PropValT` structure) within a property list. `F_ApiGetPropIndex()` is a convenience routine that makes it easier to manipulate the properties in a property list.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiGetPropIndex(F_PropValsT *pvp,
    IntT propNum);
```

Arguments

`pvp`	The property list
`propNum`	The property whose index you want to get

Returns

The index (in the property list) of the property's `F_PropValT` structure, or `FE_BadPropNum` if an error occurs.

Example

The following code gets the session property that specifies the name of the current FrameMaker product. It first gets the entire `FO_Session` property list and then uses `F_ApiGetPropIndex()` to get the index of the property.

```
. . .
IntT i;
F_PropValsT props;
props = F_ApiGetProps(0, FV_SessionId);

i = F_ApiGetPropIndex(&props, FP_ProductName);
F_ApiAlert(props.val[i].propVal.u.sval,
       FF_ALERT_CONTINUE_NOTE);
. . .
```

See also

F_ApiGetProps() and F_ApiSetProps().

F_ApiGetProps()

`F_ApiGetProps()` gets the entire property list for a specified object.

Synopsis

```
#include "fapi.h"
. . .
F_PropValsT F_ApiGetProps(F_ObjHandleT docId,
       F_ObjHandleT objId);
```

Arguments

`docId`	The ID of the session, dialog box, book, or document containing the object
`objId`	The ID of the object whose property list you want to get

Returns

An `F_PropValsT` structure that includes an array of property-value pairs.

The returned `F_PropValsT` structure references memory that is allocated by the API. Use `F_ApiDeallocatePropVals()` to free this memory when you are done with it.

If `F_ApiGetProps()` fails, the API sets the `len` field of the returned structure to `0` and assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code copies the properties of the Body paragraph to the Heading1 paragraph format. It does not affect any paragraphs that are already tagged with Heading1.

```
. . .
F_PropValsT props;
F_ObjHandleT docId, bodyFmtId, heading1Id;

/* Get IDs of document and Body and Heading1 formats. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
bodyFmtId = F_ApiGetNamedObject(docId, FO_PgfFmt, "Body");
heading1Id = F_ApiGetNamedObject(docId, FO_PgfFmt, "Heading1");
if (!bodyFmtId || !heading1Id) return;

/* Copy properties from Body format to Heading1 format. */
props = F_ApiGetProps(docId, bodyFmtId);

F_ApiSetProps(docId, heading1Id, &props);
F_ApiDeallocatePropVals(&props);
. . .
```

See also

F_ApiSetProps().

F_ApiGetPropVal()

`F_ApiGetPropVal()` queries a property of any type. If you know a property's type, it is normally easier to call an `F_ApiGet`*`PropertyType`*`()` function, such as `F_ApiGetInt()` or `F_ApiGetString()`.

Synopsis

```
#include "fapi.h"
. . .
F_PropValT F_ApiGetPropVal(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query. If the object is a session, specify `0`.
`objId`	The ID of the object whose property you want to query.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_FnNum`.

Returns

The `F_PropValT` structure for the specified property.

The returned `F_PropValT` structure references memory that is allocated by the API. Use `F_ApiDeallocatePropVal()` to free this memory when you are done with it.

If `F_ApiGetPropVal()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code determines whether strikethrough is enabled for the Emphasis character format:

```
. . .
F_PropValT prop;
F_ObjHandleT docId, charFmtId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
charFmtId = F_ApiGetNamedObject(docId, FO_CharFmt, "Emphasis");
prop = F_ApiGetPropVal(docId, charFmtId, FP_Strikethrough);
if(prop.propVal.u.ival == True)
   F_Printf(NULL,"Strikethrough is enabled for Emphasis.\n");
else
   F_Printf(NULL,"Strikethrough is not enabled.\n");
. . .
```

See also

F_ApiGetProps(), F_ApiGetTextPropVal(), and F_ApiSetPropVal().

F_ApiGetSaveDefaultParams()

`F_ApiGetSaveDefaultParams()` gets a default property list that you can use to call `F_ApiSave()`.

Synopsis

```
#include "fapi.h"
. . .
F_PropValsT F_ApiGetSaveDefaultParams();
```

Arguments

None.

Returns

A property list (an `F_PropValsT` data structure) with the properties shown in the following table. The first value listed next to each property is the value that `F_ApiGetSaveDefaultParams()` assigns to the property. The other values are values that you can change the property to.

F_ApiGetSaveDefaultParams()

Property	Meaning and Possible Values
`FS_AlertUserAboutFailure`	Specifies whether to notify user if something unusual happens during the Save operation.
	`False`: don't notify user.
	`True`: notify user.
`FS_AutoBackupOnSave`	Specifies whether to create a backup file.
	`FV_SaveUserPrefAutoBackup`: follow preference specified by the session's `FP_AutoBackup` property.
	`FV_SaveYesAutoBackup`: make a backup.
	`FV_SaveNoAutoBackup`: don't make a backup.

F_ApiGetSaveDefaultParams() Property	Meaning and Possible Values
FS_DitaGenerateComponentsAtOneLoc	While generating book with components, specifies whether individual components should be generated along with source files or along with target book file location. FV_DoUserPreference (default): Use user-specified prefences FV_DoYes: Generate at target book location FV_DoNo: Generate at source location
FS_DitaPostProcessingOnBook	Specifies that post-processing should be performed after saving DITA map. This will result in auto-numbering, generation of TOC, LOT, application of templates, and so on per the settings. FV_DoUserPreference (default): Use user-specified prefences FV_DoYes: Perform post-processing FV_DoNo: Do not perform post-processing
FS_DitaSavePdfViaBook	Specifies that while saving a DITA map to PDF, the book route should be used instead of composite route. FV_DoUserPreference (default): Use user-specified prefences FV_DoYes: Use the book route FV_DoNo: Use the composite route
FS_DontNotifyAPIClients	Specifies whether to save the file without notifying other clients. False: notify other clients. True: don't notify other clients.

F_ApiGetSaveDefaultParams() Property	Meaning and Possible Values
FS_FileType	Specifies the type of file to save to. This file type must be one that FrameMaker products save natively. Note that HTML and XML are saved via filters, and so you must specify a filter hint string via FS_SaveFileTypeHint.
	FV_SaveFmtBinary: save in Frame binary format.
	FV_SaveFmtBinary60 : save in binary format for FrameMaker 6.0
	FV_SaveFmtBinary80: This is used to save a document as 'Document 8.0', that is, FrameMaker version 8 binary document
	FV_SaveFmtBinary110: This is used to save a document as 'Document 11.0', that is, FrameMaker version 11 binary document.
	FV_SaveFmtInterchange: save as MIF.
	FV_SaveFmtInterchange70: save as MIF version 7 document.
	FV_SaveFmtInterchange110: save as MIF version 11 document.
	FV_SaveFmtPdf : save as PostScript, and then invoke Acrobat Distiller to create a PDF version of the document.This is the same as choosing PDF from the Format popup menu in the Save As dialog box.
	FV_SaveFmtViewOnly: save in View Only format
	FV_SaveFmtSgml: save in SGML format.
	FV_SaveFmtText: save in Text Only format.
	FV_SaveFmtXML: save in XML format.
	FV_SaveFmtFilter: filter on save, using FS_SaveFileTypeHint to determine the filter.
FS_FileIsInUse	Another user or session is recorded in the file's lock file.
	FV_DoCancel: Cancel the save operation
	FV_DoShowDialog: Display the File In Use dialog box
	FV_ResetLockAndContinue: Attempt to reset the file lock and save the document

F_ApiGetSaveDefaultParams() Property	Meaning and Possible Values
FS_LockCantBeReset	The user clicked Save Anyway in the File In Use dialog box, or the value of FS_FileInUse is set to FV_ResetLockAndContinue, but the lock file can't be reset. This is usually due to permissions in the lock file.
	FV_DoCancel: Cancel the save operation
	FV_DoShowDialog: Display the Cannot Lock File dialog box
	FV_DoOk:Save the document anyway
FS_ModDateChanged	The file has changed since the last time it was opened or saved in the current session. Somebody else has probably modified the file.
	FV_DoCancel: Cancel the save operation
	FV_DoShowDialog: Display the File Has Changed alert box
	FV_DoOk:Save the document anyway
FS_NumSaveParams	The available number of properties in the save script that user can set.
FS_NumSaveReturnParams	The maximum number of properties that a-save-return script can have.
FS_SaveFileNotWritable	The file permissions will not allow the file to be saved.
	FV_DoCancel: Cancel the save operation
	FV_DoShowDialog: Display the Cannot Lock FIle alert box
FS_SaveFileTypeHint	If FS_FileType is FV_SaveFmtFilter, this string enables the FrameMaker product to call the correct filter. For example, use 0001ADBEHTML to save as HTML or 0001ADBEXML to save as XML. For information on hint string syntax, see Syntax of FP_ImportHint strings or Syntax of FP_ImportHint strings.
	NULL.

F_ApiGetSaveDefaultParams() Property	Meaning and Possible Values
FS_MakePageCount	Specifies how to round the page count.
	FV_UseCurrentSetting: use default specified by the document property, FP_PageRounding.
	FV_DontChangePageCount: leave pages as is.
	FV_MakePageCountEven: with odd number of pages, add a page to end of document.
	FV_MakePageCountOdd: with even number of pages, add a page to end of document.
	FV_DeleteEmptyPages: remove extra pages at end of document.
FS_RetainNameStripe	Specifies whether to change the name in document title bar to the name the file is saved to.
	False: change name in title bar to the name the file is saved to.
	True: do not change name in title bar.
FS_SaveAsModeName	Specifies where to get filename if FS_SaveMode set to FV_ModeSaveAs.
	FV_SaveAsNameProvided: save under the filename specified in saveAsName parameter of F_ApiSave().
	FV_SaveAsUseFileName: save as name shown on document title bar.
	FV_SaveAsNameAskUser: prompt user for name.
FS_SaveMode	Specifies whether to use Save or Save As mode.
	FV_ModeSaveAs: use Save As mode.
	FV_ModeSave: use Save mode.
FS_SaveTextExtraBlank LineAtEOP	Specifies whether to add an extra line at the end of each paragraph if the file is being saved as Text Only.
	False: don't add an extra line.
	True: add an extra line.

F_ApiGetSaveDefaultParams() Property	Meaning and Possible Values
FS_SaveTextTblSetting	Specifies how to deal with tables if the file is being saved as Text Only.
	FV_SaveTblUserPref: use setting last specified in Save as Text dialog box.
	FV_SaveTblRowsAsPgfs: save each table cell as a paragraph row-by-row.
	FV_SaveTblColsAsPgfs: save each table cell as a paragraph column-by-column.
	FV_SaveSkipTbls: omit tables from a Text Only file.
	FV_SaveTextTblCellSeparator: the character to write as a cell separator in the text file.
	FV_SaveTextTblRowColumnSeparator: the character to write as a row or column separator in the text file.
FS_FrameMaker+SGMLSaveApplication	Retained for compatibility. Use FS_StructuredSaveApplication
FS_ShowSaveTextDialog	Specifies whether to display dialog box if the file is being saved in Text Only format.
	False: don't display dialog box.
	True: display dialog box asking user whether to put paragraph returns at the end of each line.
FS_UpdateFRVList	Specifies whether the file will be added to the list of files recently visited that appears in the File menu. This is set to False by default.
	False: don't add the file to the list.
	True: add the file to the list.

If F_ApiGetSaveDefaultParams() fails, the API sets the len field of the returned structure to 0.

The returned F_PropValsT structure references memory that is allocated by the API. Use F_ApiDeallocatePropVals() to free this memory when you are done with it.

Example

The following code uses `F_ApiGetSaveDefaultParams()` to get a default `F_ApiSave()` property list. It modifies the property list to instruct `F_ApiSave()` to save `/tmp/my.doc` as a View Only document named `/tmp/viewonly.doc`.

```
. . .
IntT i;
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT mydocId, viewonlydocId;

/* Open my.doc. */
mydocId = F_ApiSimpleOpen("/tmp/my.doc", False);

/* Get default script. */
params = F_ApiGetSaveDefaultParams();

/* Get index of FS_FileType property. Then set it. */
i = F_ApiGetPropIndex(&params, FS_FileType);
params.val[i].propVal.u.ival =
                  FV_SaveFmtViewOnly;

/* Save document, using modified default property list. */
viewonlydocId = F_ApiSave(mydocId, "/tmp/viewonly.doc",
           &params, &returnParamsp);

/* Deallocate scripts to save memory. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
. . .
```

See also

F_ApiSave().

F_ApiGetString()

`F_ApiGetString()` queries a string (`StringT`) property.

Synopsis

```
#include "fapi.h"
. . .
StringT F_ApiGetString(F_ObjHandleT docId,
       F_ObjHandleT objId,
       IntT propNum);
```

Arguments

`docId`	The ID of the document, dialog box, book, or session containing the object whose property you want to query. If the object is a session, specify `0`.
`objId`	The ID of the object whose property you want to query.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_Name`.

Returns

The string (`StringT`) for the specified property, or `NULL` if an error occurs.

IMPORTANT: *Use* `F_Free()` *to free the string returned by* `F_ApiGetString()` *when you are done with it.*

If `F_ApiGetString()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following example gets the name of the FrameMaker product currently running:

```
. . .
StringT productName;
productName = F_ApiGetString(0, FV_SessionId, FP_ProductName);
F_ApiAlert(productName, FF_ALERT_CONTINUE_NOTE);

F_Free(productName);
. . .
```

See also

F_ApiSetString().

F_ApiGetStrings()

`F_ApiGetStrings()` queries a multiple-strings (`F_StringsT`) property. It is useful for retrieving the list of font families in a session, the words in the document dictionary, or the list of marker names in a document.

Synopsis

```
#include "fapi.h"
. . .
F_StringsT F_ApiGetStrings(F_ObjHandleT docId,
       F_ObjHandleT objId,
       IntT propNum);
```

Arguments

`docId`	The ID of the document, dialog box, book, or session containing the object whose property you want to query. If the object is a session, specify `0`.
`objId`	The ID of the object whose property you want to query.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_Dictionary` or `FP_MarkerNames`.

Returns

An `F_StringsT` structure that contains a pointer to the array of strings.

The returned `F_StringsT` structure references memory that is allocated by the API. Use `F_ApiDeallocateStrings()` to free this memory when you are done with it.

If `F_ApiGetStrings()` fails, the API sets the `len` field of the returned structure to `0` and assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code prints the list of font family names for the current session to the console or error log window:

```
. . .
#include "futils.h"
IntT i;
F_StringsT strings;

strings = F_ApiGetStrings(0, FV_SessionId, FP_FontFamilyNames);

/* Display font family names.*/
for (i=0; i<strings.len; i++)
   F_Printf(NULL,"Family %d: %s\n", i, strings.val[i]);
F_ApiDeallocateStrings(&strings);
. . .
```

See also

F_ApiSetStrings().

F_ApiGetSupportedEncodings()

`F_ApiGetSupportedEncodings()` returns the font encodings supported for the current session.

Synopsis

```
#include "fapi.h"
. . .
F_StringsT F_ApiGetSupportedEncodings();
```

Returns

A list of all the encodings supported for the current session. The following strings indicate encoding for text fonts:

Value	Means
`FrameRoman`	Roman text
`JISX0208.ShiftJIS`	Japanese text
`BIG5`	Traditional Chinese text
`GB2312-80.EUC`	Simplified Chinese text
`KSC5601-1992`	Korean text

Non-text fonts may return `FrameRoman`, or they may return the family name of the font. For example, on some platforms the encoding for the Symbol font family is indicated by the string `Symbol`.

Example

The following code prints the list of supported encodings to the console:

```
. . .
#include "futils.h"
#include "fstrings.h"
. . .
IntT i;
F_StringsT encodings;

encodings = F_ApiGetSupportedEncodings();
for (i=0; i < encodings.len; i++)
   F_Printf(NULL,"Encoding %d: %s\n", i, encodings.val[i]);
F_ApiDeallocateStrings(&encodings);
. . .
```

F_ApiGetTabs()

`F_ApiGetTabs()` queries a tabs (`F_TabsT`) property. The tabs are returned in left-to-right order.

Synopsis

```
#include "fapi.h"
. . .
F_TabsT F_ApiGetTabs(F_ObjHandleT docId,
   F_ObjHandleT objId,
   IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query
`objId`	The ID of the object whose property you want to query
`propNum`	The property to query (for example, `FP_Tabs`)

Returns

An `F_TabsT` structure containing the tabs.

`F_TabsT` is defined as:

```
typedef struct {
   UIntT len; /* The number of tabs in val */
   F_TabT *val; /* Structures that describe the tabs */
} F_TabsT;
```

The `F_TabT` structure is defined as:

```
typedef struct {
   MetricT x; /* Offset from paragraph's left margin */
   UCharT type; /* Constant indicating tab, e.g. FV_TAB_RIGHT */
   StringT leader; /* Characters before tab, e.g. "." */
   UCharT decimal; /* Character for decimal tab, e.g. "." */
} F_TabT;
```

`F_TabT.type` can be one of the following.

Type constant	Tab type
`FV_TAB_LEFT`	Left tab
`FV_TAB_CENTER`	Center tab
`FV_TAB_RIGHT`	Right tab
`FV_TAB_DECIMAL`	Decimal tab
`FV_TAB_RELATIVE_LEFT`	Relative left tab (allowed only for format change lists)
`FV_TAB_RELATIVE_CENTER`	Relative center tab (allowed only for format change lists)
`FV_TAB_RELATIVE_RIGHT`	Relative right tab (allowed only for format change lists)
`FV_TAB_RELATIVE_DECIMAL`	Relative decimal tab (allowed only for format change lists)

If `F_ApiGetTabs()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

The returned `F_TabsT` structure references memory that is allocated by the API. Use `F_ApiDeallocateTabs()` to free this memory when you are done with it.

Example

The following example displays information about tabs in the paragraph containing the insertion point:

```
. . .
#include "futils.h"
#define in (RealT) (65536*72)
F_ObjHandleT docId;
IntT i;
F_TabsT tabs;
F_TextRangeT tr;
UCharT msg[256];

/* Get the active document and its current text selection.*/
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Get the tabs for paragraph at end of text selection.*/
tabs = F_ApiGetTabs(docId, tr.end.objId, FP_Tabs);

/* Iterate through tabs, displaying their characteristics.*/
for (i=0; i<tabs.len; i++)
   {
   F_Sprintf(msg,
          "Tab %d: type=%d, x=%2.2f, decimal=%c, leader=%s",
          i,
          tabs.val[i].type,
          tabs.val[i].x/in,
          tabs.val[i].decimal,
          tabs.val[i].leader);
   F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
   }
F_ApiDeallocateTabs(&tabs);
. . .
```

See also

F_ApiSetTabs().

F_ApiGetText()

`F_ApiGetText()` gets the text from the following types of objects:

- `FO_Cell`
- `FO_Element`
- `FO_Flow`
- `FO_Fn`
- `FO_Pgf`
- `FO_SubCol`
- `FO_TextFrame`
- `FO_TextLine`
- `FO_TiApiClient`
- `FO_TiFlow`
- `FO_TiText`
- `FO_TiTextTable`
- `FO_Var`
- `FO_XRef`

`F_ApiGetText()` returns a structure containing an array of text items. Each text item contains either a string of text, the ID of an object that appears within the text (such as a table or an anchored frame), an indicator that the text properties have changed, or the ID of an object that organizes the text (such as a paragraph or a text column). For more information on text items and how text is represented in FrameMaker products, see "How the API represents text" on page 112 of the *FDK Programmer's Guide*.

Always deallocate the memory used by the `F_TextItemsT` structure returned by `F_ApiGetText()` when you are through using it. The API provides a convenience function, named `F_ApiDeallocateTextItems()`, that does this for you.

Synopsis

```
#include "fapi.h"
. . .
F_TextItemsT F_ApiGetText(F_ObjHandleT docId,
       F_ObjHandleT objId,
       IntT flags);
```

Arguments

`docId`	The ID of the document containing the object that contains the text.
`objId`	The ID of the object containing the text you want to get.
`flags`	Bit flags that specify the type of text items to retrieve. To get specific types of text items, OR the constants that represent them (for example, use a bitwise OR to combine `FTI_FlowBegin` and `FTI_String`) into `flags`. To get all types of text items, specify `-1`. For a complete list of the constants that represent text item types, see the table below.

Returns

An `F_TextItemsT` structure containing the array of text items of the requested types in the paragraph.

`F_TextItemsT` is defined as:

```
typedef struct {
    UIntT len;      /* Number of text items. */
    F_TextItemT *val; /* Array of text items. */
} F_TextItemsT;
```

The `F_TextItemT` structure, which specifies an individual text item, is defined as:

```
typedef struct{
   IntT offset; /* From start of object */
   IntT dataType;  /* Text item type, e.g. FTI_String. */
   union {
      StringT sdata; /* A string if the type is FTI_String */
      F_ObjHandleT idata; /* ID if type is an anchor. */
   } u;
} F_TextItemT;
```

`F_TextItemT.dataType` can be one of the following constants:

Text item type (dataType)	What the text item represents	Text item data
`FTI_CharPropsChange`	A change in the text properties	Flags indicating which properties have changed (see the table below)
`FTI_ElementBegin`	The beginning of a container structural element	ID of an `FO_Element`
`FTI_ElementEnd`	The end of a container structural element	ID of an `FO_Element`
`FTI_ElemPrefixBegin`	The beginning of an element's prefix	ID of an `FO_Element`
`FTI_ElemPrefixEnd`	The end of an element's prefix	ID of an `FO_Element`
`FTI_ElemSuffixBegin`	The beginning of an element's suffix	ID of an `FO_Element`
`FTI_ElemSuffixEnd`	The end of an element's suffix	ID of an `FO_Element`
`FTI_FlowBegin`	The beginning of a flow	ID of an `FO_Flow`
`FTI_FlowEnd`	The end of a flow	ID of an `FO_Flow`
`FTI_FnAnchor`	A footnote	ID of an `FO_Fn`
`FTI_FrameAnchor`	An anchored frame	ID of an `FO_AFrame`
`FTI_LineBegin`	The beginning of a line	Nothing
`FTI_LineEnd`	The end of a line and the line end type	If the line end is a normal line end, `0`; if it is a forced line end, the `FTI_HardLineEnd` flag is set; if it is a hyphen line end, the `FTI_HyphenLineEnd` flag is set
`FTI_MarkerAnchor`	A marker	ID of an `FO_Marker`
`FTI_PageBegin`	The beginning of a page	ID of an `FO_Page`
`FTI_PageEnd`	The end of a page	ID of an `FO_Page`
`FTI_PgfBegin`	The beginning of a paragraph	ID of an `FO_Pgf`
`FTI_PgfEnd`	The end of a paragraph	ID of an `FO_Pgf`

Text item type (dataType)	What the text item represents	Text item data
FTI_String	A string of characters with the same condition and character format	A character string
FTI_SubColBegin	The beginning of a column	ID of an FO_SubCol
FTI_SubColEnd	The end of a column	ID of an FO_SubCol
FTI_TblAnchor	A table	ID of an FO_Tbl
FTI_TextFrameBegin	The beginning of a text frame	ID of an FO_TextFrame
FTI_TextFrameEnd	The end of a text frame	ID of an FO_TextFrame
FTI_TextInsetBegin	The beginning of a text inset	ID of an FO_TiApiClient, FO_TiFlow, FO_TiText, or FO_TiTextTable
FTI_TextInsetEnd	The end of a text inset	ID of an FO_TiApiClient, FO_TiFlow, FO_TiText, or FO_TiTextTable
FTI_TextObjId	The object that the offsets of all the text items are relative to	ID of an FO_Pgf or FO_TextLine
FTI_VarBegin	The beginning of a variable	ID of an FO_Var
FTI_VarEnd	The end of a variable	ID of an FO_Var
FTI_XRefBegin	The beginning of a cross-reference	ID of an FO_XRef
FTI_XRefEnd	The end of a cross-reference	ID of an FO_XRef

The following table lists the bit flags that a client can bitwise AND with the `idata` field of an `FTI_CharPropsChange` text item. For example, to determine if the font family changed, bitwise AND the `FTF_FAMILY` flag with the `idata` field.

Flag	Meaning
`FTF_ALL`	OR of all the flags listed above.
`FTF_ANGLE`	The font angle has changed.
`FTF_CAPITALIZATION`	The capitalization has changed.
`FTF_CHANGEBAR`	The change bars have changed.
`FTF_CHARTAG`	The Character Catalog format has changed.
`FTF_COLOR`	The color has changed.
`FTF_CONDITIONTAG`	The condition tag has changed.
`FTF_ENCODING`	The text encoding has changed.
`FTF_FAMILY`	The font family has changed.
`FTF_IIF`	An internal flag having to do with asian text. input. If there is a non-zero value for this flag, a front end processor is controlling that text; you should not modify the associated text item.
`FTF_KERNX`	The kern-x characteristic has changed.
`FTF_KERNY`	The kern-y characteristic has changed.
`FTF_LANGUAGE`	Character language has changed
`FTF_OUTLINE`	The outline characteristic has changed.
`FTF_OVERLINE`	The overline characteristic has changed.
`FTF_PAIRKERN`	The pair kerning has changed.
`FTF_POSITION`	The character position has changed.
`FTF_SHADOW`	The shadow characteristic has changed.
`FTF_SIZE`	The font size has changed.
`FTF_SPREAD`	The font spread has changed.
`FTF_STRETCH`	Font stretch value has changed
`FTF_STRIKETHROUGH`	The strikethrough characteristic has changed.
`FTF_TSUME`	Tsume setting has changed
`FTF_UNDERLINING`	The underlining has changed.

Flag	Meaning
FTF_VARIATION	The font variation has changed.
FTF_WEIGHT	The font weight has changed.

If F_ApiGetText() fails, the API sets the len field of the returned structure to 0 and assigns one of the following values to FA_errno.

FA_errno value	Meaning
FE_BadDocId	Invalid document ID
FE_BadObjId	Invalid object ID
FE_NotTextObject	Object specified for the text location is not an object that contains text

If you call F_ApiGetText() for a structural element (FO_Element object), the returned information depends on the type of element, as shown in the following table:

Element's FP_ElementType value	Information returned by F_ApiGetText()
FV_FO_CONTAINER	
FV_FO_SYS_VAR	
FV_FO_XREF	the cross-reference.
FV_FO_FOOTNOTE	All the text items from the beginning to the end of the footnote.
FV_FO_TBL_TITLE	All the text items from the beginning to the end of the table title.
FV_FO_TBL_CELL	All the text items from the beginning to the end of the cell.

Element's FP_ElementType value	Information returned by F_ApiGetText()
FV_FO_TBL_HEADING	Nothing. `F_ApiGetText()` fails.
FV_FO_TBL_BODY	
FV_FO_TBL_FOOTING	
FV_FO_MARKER	
FV_FO_TBL	
FV_FO_GRAPHIC	
FV_FO_EQN	
FV_FO_TBL_ROW	

The returned `F_TextItemsT` structure references memory that is allocated by the API. Use `F_ApiDeallocateTextItems()` to free this memory when you are done with it.

Example

The following code displays the number of lines in the paragraph containing the insertion point (or the beginning of the current text selection):

```
. . .
#include "futils.h"
F_ObjHandleT docId;
F_TextRangeT tr;
F_TextItemsT tis;
UCharT msg[256];

/* Get active document and the selection or insertion point. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Get all the line-end text items in the paragraph. */
tis = F_ApiGetText(docId, tr.beg.objId, FTI_LineEnd);

F_Sprintf(msg, "Number of lines is %d.", tis.len);
F_ApiAlert(msg, FF_ALERT_CONTINUE_WARN);
F_ApiDeallocateTextItems(&tis);
. . .
```

See also

F_ApiAddText(), F_ApiDeallocateStructureType(), and F_ApiDeleteText().

F_ApiGetText2()

`F_ApiGetText2()` gets the text from all the objects available to `F_ApiGetText()`, as well as an extended set of document objects that includes the following:

- `FO_Rubi`

IMPORTANT: *This function takes two* `flags` *arguments, thus extending the number of text item types you can retrieve. Except for the additional* `flags` *argument, this function is the same as* `F_ApiGetText()`*, described on page 249. Note that* `F_ApiGetText()` *still works; you just can't use it to access the extended set of text item types.*

`F_ApiGetText2()` returns a structure containing an array of text items. Each text item contains either a string of text, the ID of an object that appears within the text (such as a table or an anchored frame), an indicator that the text properties have changed, or the ID of an object that organizes the text (such as a paragraph or a text column). For more information on text items and how text is represented in FrameMaker products, see "How the API represents text" on page 112 of the *FDK Programmer's Guide*.

Always deallocate the memory used by the `F_TextItemsT` structure returned by `F_ApiGetText2()` when you are through using it. The API provides a convenience function, named `F_ApiDeallocateTextItems()`, that does this for you.

Synopsis

```
#include "fapi.h"
. . .
F_TextItemsT F_ApiGetText2(F_ObjHandleT docId,
       F_ObjHandleT objId,
       IntT flags,
       IntT flags2);
```

Arguments

`docId`	The ID of the document containing the object that contains the text.
`objId`	The ID of the object containing the text you want to get.
`flags`	Bit flags for the base set text item types that of text items to retrieve. For a complete list of the base set of constants that represent text item types, see F_ApiGetText().
`flags2`	Bit flags for the extended set of text item types that specify the type of text items to retrieve. To get specific types of text items from the extended set, OR the constants that represent them (for example, use a bitwise OR to combine `FTI2_RubiTextBegin` and `FTI2_RubiTextEnd`) into `flags2`. To get all types of the extended set of text items, specify `-1`. For a complete list of the constants that represent the extended set of text item types, see the table below.

Returns

An `F_TextItemsT` structure containing the array of text items of the requested types in the paragraph.

`F_TextItemsT` is defined as:

```
typedef struct {
    UIntT len;      /* Number of text items. */
    F_TextItemT *val; /* Array of text items. */
} F_TextItemsT;
```

The `F_TextItemT` structure, which specifies an individual text item, is defined as:

```
typedef struct{
   IntT offset; /* From start of object */
   IntT dataType;  /* Text item type, e.g. FTI_String. */
   union {
      StringT sdata; /* A string if the type is FTI_String */
      F_ObjHandleT idata; /* ID if type is an anchor. */
   } u;
} F_TextItemT;
```

`F_TextItemT.dataType` can be one of the following constants:

Text item type (dataType)	What the text item represents	Text item data
`FTI2_RubiTextBegin`	The beginning of rubi text (and implicitly, the beginning of rubi text	ID of the `FO_Rubi` object for the rubi composite that contains the rubi text
`FTI2_RubiTextEnd`	The end of rubi text	ID of the `FO_Rubi` object for the rubi composite that contains the rubi text
`FTI2_RubiComposite Begin`	The beginning of a rubi composite (and implicitly, the beginning of oyamoji text)	ID of an `FO_Rubi` object
`FTI2_RubiComposite End`	The end of a rubi composite	ID of an `FO_Rubi` object

If `F_ApiGetText2()` fails, the API sets the `len` field of the returned structure to `0` and assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_NotTextObject`	Object specified for the text location is not an object that contains text

The returned `F_TextItemsT` structure references memory that is allocated by the API. Use `F_ApiDeallocateTextItems()` to free this memory when you are done with it.

Example

The following code displays the number of rubi compositesin the paragraph containing the insertion point (or the beginning of the current text selection):

```
. . .
#include "futils.h"
. . .
F_ObjHandleT docId;
F_TextRangeT tr;
F_TextItemsT tis;
UCharT msg[256];

/* Get active document and the selection or insertion point. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Get all the rubi composite begin text items in the paragraph.
*/
tis = F_ApiGetText2(docId, tr.beg.objId, 0,
                    FTI2_RubiCompositeBegin);

F_Sprintf(msg, "Number of rubi composites is %d.", tis.len);
F_ApiAlert(msg, FF_ALERT_CONTINUE_WARN);
F_ApiDeallocateTextItems(&tis);
. . .
```

See also

F_ApiAddText(), F_ApiDeallocateStructureType(), and F_ApiDeleteText().

F_ApiGetTextForRange()

`F_ApiGetTextForRange()` gets the text for a specified text range. It provides the same parameters and functionality as `F_ApiGetText()`, except it allows you to specify a text range instead of an object.

Synopsis

```
#include "fapi.h"
. . .
F_TextItemsT F_ApiGetTextForRange(F_ObjHandleT docId,
    F_TextRangeT *tr,
    IntT flags);
```

Arguments

`docId`	The ID of the document containing the text range that contains the text.
`tr`	The text range containing the text you want to get. For information on specifying text ranges, see "Getting and setting the insertion point or text selection" on page 321 of the *FDK Programmer's Guide*.
`flags`	Bit flags that specify the type of text items to retrieve. To get specific types of text items, OR the constants that represent them (for example, `FTI_FlowBegin` and `FTI_String`) into `flags`. To get all types of text items, specify `-1`. For a complete list of the constants that represent text item types, see F_ApiGetText().

Returns

An `F_TextItemsT` structure containing the array of text items of the requested types in the paragraph.

If `F_ApiGetTextForRange()` fails, the API sets the `len` field of the returned structure to `0` and assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_NotTextObject`	Object specified for the text range is not an object that contains text
`FE_BadRange`	Specified text range is invalid
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified paragraph or text line

The returned `F_TextItemsT` structure references memory that is allocated by the API. Use `F_ApiDeallocateTextItems()` to free this memory when you are done with it.

Example

The following code prints the selected text in the active document:

```
. . .
F_ObjHandleT docId;
F_TextRangeT tr;
F_TextItemsT tis;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);

/* If there's just an insertion point, no text is selected. */
if(tr.beg.objId == tr.end.objId
   && tr.beg.offset == tr.end.offset) return;

tis = F_ApiGetTextForRange(docId, &tr, FTI_String);

/* Traverse text items and print strings and line ends. */
for (i=0; i<tis.len; i++)
{
   F_Printf(NULL,"%s", tis.val[i].u.sdata);
}
F_ApiDeallocateTextItems(&tis);
. . .
```

See also

F_ApiAddText(), F_ApiDeallocateStructureType(), and F_ApiGetText().

F_ApiGetTextForRange2()

`F_ApiGetTextForRange2()` gets the text for a specified text range. It provides the same parameters and functionality as `F_ApiGetText2()`, except it allows you to specify a text range instead of an object.

Synopsis

```
#include "fapi.h"
. . .
F_TextItemsT F_ApiGetTextForRange(F_ObjHandleT docId,
    F_TextRangeT *tr,
    IntT flags
    IntT flags2);
```

Arguments

`docId`	The ID of the document containing the text range that contains the text.
`tr`	The text range containing the text you want to get. For information on specifying text ranges, see "Getting and setting the insertion point or text selection" on page 321 of the *FDK Programmer's Guide*.
`flags`	Bit flags for the base set text item types that of text items to retrieve. For a complete list of the base set of constants that represent text item types, see F_ApiGetText().
`flags2`	Bit flags for the extended set of text item types that specify the type of text items to retrieve. To get specific types of text items from the extended set, OR the constants that represent them (for example, use a bitwise OR to combine `FTI2_RubiTextBegin` and `FTI2_RubiTextEnd`) into `flags2`. To get all types of the extended set of text items, specify `-1`. For a complete list of the constants that represent the extended set of text item types, see the table below.

Returns

An `F_TextItemsT` structure containing the array of text items of the requested types in the paragraph.

If `F_ApiGetTextForRange()` fails, the API sets the `len` field of the returned structure to `0` and assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_NotTextObject`	Object specified for the text range is not an object that contains text
`FE_BadRange`	Specified text range is invalid
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified paragraph or text line

The returned `F_TextItemsT` structure references memory that is allocated by the API. Use `F_ApiDeallocateTextItems()` to free this memory when you are done with it.

Example

The following code prints the selected text in the active document:

```
. . .
F_ObjHandleT docId;
F_TextRangeT tr;
F_TextItemsT tis;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);

/* If there's just an insertion point, no text is selected. */
if(tr.beg.objId == tr.end.objId
   && tr.beg.offset == tr.end.offset) return;

/* Get all the rubi composite begin text items in the selection.
*/
tis = F_ApiGetTextForRange2(docId, &tr, 0,
                    FTI2_RubiCompositeBegin);

F_Sprintf(msg, "Number of rubi composites is %d.", tis.len);
F_ApiAlert(msg, FF_ALERT_CONTINUE_WARN);
F_ApiDeallocateTextItems(&tis);
. . .
```

See also

F_ApiAddText(), F_ApiDeallocateStructureType(), and F_ApiGetText().

F_ApiGetTextLoc()

`F_ApiGetTextLoc()` gets a text location (`F_TextLocT`) property.

Synopsis

```
#include "fapi.h"
. . .
F_TextLocT F_ApiGetTextLoc(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query
`objId`	The ID of the object whose property you want to query
`propNum`	The property to query

Returns

An `F_TextLocT` structure specifying a text location. The `F_TextLocT` structure is defined as:

```
typedef struct{
    F_ObjHandleT objId; /* Object containing text */
    IntT offset; /* Characters from start of object */
} F_TextLocT;
```

If `F_ApiGetTextLoc()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code moves the insertion point to the first marker in the active document's list of markers:

```
. . .
F_ObjHandleT docId, mrkrId;
F_TextLocT tl;
F_TextRangeT tr;

/* Get IDs of active document and the first marker. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
mrkrId = F_ApiGetId(FV_SessionId, docId, FP_FirstMarkerInDoc);

tl = F_ApiGetTextLoc(docId, mrkrId, FP_TextLoc);

if (tl.objId == 0) return;
tr.end.objId = tr.beg.objId = tl.objId;
tr.end.offset = tr.beg.offset = tl.offset;

F_ApiSetTextRange(FV_SessionId, docId, FP_TextSelection, &tr);
. . .
```

See also

F_ApiSetTextRange().

F_ApiGetTextProps()

`F_ApiGetTextProps()` gets the text properties (such as the format tag, font family and size, and conditions) for a location in text. Because the text properties can be different for each character, you can only get the text properties for an individual location in text.

Synopsis

```
#include "fapi.h"
. . .
F_PropValsT F_ApiGetTextProps(F_ObjHandleT docId,
    F_TextLocT *textLocp);
```

Arguments

`docId`	The ID of the document containing the text location.
`textLocp`	The text location of the character that you want to get text properties for. The returned properties are the properties that apply to the character to the right of the specified location. For information on specifying text locations, see "Getting and setting the insertion point or text selection" on page 321 of the *FDK Programmer's Guide*.

Returns

The text property list (an `F_PropValsT` structure) for the text location.

If `F_ApiGetTextProps()` fails, the API sets the `len` field of the returned structure to `0` and assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_NotTextObject`	Object specified for the text location is not an object that contains text
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified paragraph or text line
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

The returned `F_PropValsT` structure references memory that is allocated by the API. Use `F_ApiDeallocatePropVals()` to free this memory when you are done with it.

Example

The following code applies the text properties of the first character in the current selection to the rest of the selection:

```
. . .
#include "futils.h"
F_TextRangeT tr;
F_PropValsT props;
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Get insertion point or text selection and its properties. */
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (tr.beg.objId == 0) return;
props = F_ApiGetTextProps(docId, &tr.beg);

/* Apply the properties to the entire selection. */
F_ApiSetTextProps(docId, &tr, &props);

F_ApiDeallocatePropVals(&props);
. . .
```

See also

F_ApiSetTextProps().

F_ApiGetTextPropVal()

`F_ApiGetTextPropVal()` gets a text property (such as the format tag, font family and size, or conditions) for a location in text. Because a text property can be different for each character, you can get it for only one location in text at a time.

Synopsis

```
#include "fapi.h"
. . .
F_PropValT F_ApiGetTextPropVal(F_ObjHandleT docId,
    F_TextLocT *textLocp,
    IntT propNum);
```

Arguments

`docId`	The ID of the document containing the text location.
`textLocp`	The text location of the character that you want to get the text property for. The returned property applies to the character to the right of this location.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_FontFamily`.

Returns

The `F_PropValT` structure for the specified property.

The returned `F_PropValT` structure references memory that is allocated by the API. Use `F_ApiDeallocatePropVal()` to free this memory when you are done with it.

If `F_ApiGetTextPropVal()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_NotTextObject`	Object specified for the text location is not an object that contains text
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified paragraph or text line
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code prints the font family of the character to the right of the insertion point:

```
. . .
F_TextRangeT tr;
F_PropValT prop;
F_ObjHandleT docId;
F_StringsT families;

/* Get the current insertion point. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(!tr.beg.objId) return;

/* Get list of font families in the session. */
families = F_ApiGetStrings(0, FV_SessionId, FP_FontFamilyNames);

/* Get the index of the font family (in the list of families)
** of the character to the right of the insertion point.
*/
prop = F_ApiGetTextPropVal(docId, &tr.end, FP_FontFamily);

F_Printf(NULL,"The font family is %s.\n",
          families.val[prop.propVal.u.ival]);
. . .
```

See also

F_ApiGetTextProps() and F_ApiGetTextVal().

F_ApiGetTextRange()

`F_ApiGetTextRange()` gets a text range property. It is useful for getting the insertion point or text selection in a document.

Synopsis

```
#include "fapi.h"
. . .
F_TextRangeT F_ApiGetTextRange(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query
`objId`	The ID of the object whose property you want to query
`propNum`	The property to query (for example, `FP_TextSelection`)

Returns

An `F_TextRangeT` structure specifying a text range. The `F_TextRangeT` structure is defined as:

```
typedef struct {
   F_TextLocT beg; /* The beginning of the range */
   F_TextLocT end; /* The end of the range */
} F_TextRangeT;
```

The `F_TextLocT` structure is defined as:

```
typedef struct{
   F_ObjHandleT objId; /* Object containing text. */
   IntT offset; /* Characters from beginning */
} F_TextLocT;
```

If `F_ApiGetTextRange()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

See F_ApiSetTextRange().

See also

F_ApiSetTextRange().

F_ApiGetTextVal()

`F_ApiGetTextVal()` gets the value of a specified text property, which can be of any type.

Synopsis

```
#include "fapi.h"
. . .
F_TypedValT F_ApiGetTextVal(F_ObjHandleT docId,
    F_TextLocT *textLocp,
    IntT propNum);
```

Arguments

`docId`	The ID of the document containing the text location.
`textLocp`	The text location of the character that you want to get the text property for. The returned property applies to the character to the right of this location.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_FontFamily`.

Returns

An `F_TypedValT` structure containing the value of the specified property.

If `F_ApiGetTextVal()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_NotTextObject`	Object specified for the text location is not an object that contains text
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified paragraph or text line
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code prints the name of the character format applied to the character to the right of the insertion point.

```
. . .
F_TextRangeT tr;
F_TypedValT val;
F_ObjHandleT docId;

/* Get the current insertion point. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(!tr.beg.objId) return;

val = F_ApiGetTextVal(docId, &tr.end, FP_CharTag);
F_Printf(NULL,"The character tag is %s.\n", val.u.sval);
. . .
```

See also

F_ApiGetTextProps() and F_ApiGetTextPropVal().

F_ApiGetUBytesByName()

`F_ApiGetUBytesByName()` queries an unsigned bytes (`F_UBytesT`) inset facet. The standard facets, EPSI and FrameImage, are examples of unsigned bytes facets.

If a facet contains a large amount of data, `F_ApiGetUBytesByName()` only gets a portion of the data each time you call it. To query a facet that contains a large amount of data, call `F_ApiGetUBytesByName()` repeatedly until you have retrieved all the data (that is, until `F_UBytesT.len` is `0`).

`F_ApiGetUBytesByName()` and other `F_ApiGet ByName()` functions use a transaction model to query properties. After you have finished a series of queries, you must commit the transaction by using `F_ApiGetIntByName()` to query a facet named `""`.

Synopsis

```
#include "fapi.h"
. . .
F_UBytesT F_ApiGetUBytesByName(F_ObjHandleT docId,
       F_ObjHandleT objId,
       StringT propName);
```

Arguments

`docId`	The ID of the document containing the inset whose facet you want to query
`objId`	The ID of the inset whose facet you want to query
`propName`	The name of the facet to query

Returns

An `F_UBytesT` structure that contains a portion of the facet's data.

`F_UBytesT` is defined as:

```
typedef struct {
   UIntT len; /* The number of unsigned bytes */
   UByteT *val; /* The facet data */
} F_UBytesT;
```

The returned `F_UBytesT` structure references memory that is allocated by the API. Use `F_ApiDeallocateUBytes()` to free this memory when you are done with it.

If `F_ApiGetUBytesByName()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property name is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code gets all the bytes in the `my.facet` facet of an inset:

```
. . .
F_ObjHandleT docId, insetId;
F_UBytesT aUBytes;
IntT n;

do {
    aUBytes = F_ApiGetUBytesByName(docId, insetId, "my.facet");
    n = aUBytes.len;
    if(n>0)
    {
        /* Code to do something with aUBytes goes here. */
    }
} while (n > 0);
F_ApiGetIntByName(docId, insetId, ""); /* Commit transaction. */
F_ApiDeallocateUBytes(&aUBytes);
. . .
```

See also

F_ApiSetUBytesByName().

F_ApiGetUniqueObject()

`F_ApiGetUniqueObject()` gets the ID of an object from its persistent unique identifier (UID). FrameMaker products assign a UID to each object in a document or book that isn't identified by a name. The UID, unlike the object's ID, does not change from one session to another.

IMPORTANT: *When you copy an object to the clipboard and paste it, the FrameMaker product changes the UID. This also happens when you hide and show conditional text.*

Synopsis

```
F_ObjHandleT F_ApiGetUniqueObject(F_ObjHandleT docId,
    IntT objType,
    IntT unique);
```

Arguments

`docId`	The ID of the document containing the object
`objType`	The type of object (for example, `FO_Pgf`)
`unique`	The object's UID

Returns

The ID of the object, or `0` if an error occurs.

If `F_ApiGetUniqueObject()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_TypeUnNamed`	Objects of the specified type aren't identified by UIDs
`FE_NameNotFound`	Object with specified UID couldn't be found

Example

The following code will build a client gets and saves the location of the current insertion point (or the beginning of the current text selection) when the user chooses a menu item named Save Position. If the user subsequently chooses the Return to Previous Position menu item, the client scrolls to the saved insertion point, even if the document has been exited and reopened in the meantime.

```
#include "fapi.h"
#define SAVE 1
#define LAST 2

IntT pgfUID;
F_TextRangeT tr;

VoidT F_ApiInitialize(initialization)
       IntT initialization;
{
   F_ObjHandleT menuBarId, menuId;

   menuBarId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                           "!MakerMainMenu");
   menuId = F_ApiDefineAndAddMenu(menuBarId, "APIMenu", "API");
   F_ApiDefineAndAddCommand(SAVE, menuId,"SaveLastPosCmd",
      "Save position","");
   F_ApiDefineAndAddCommand(LAST, menuId,"ReturnToLastPosCmd",
      "Return to previous position","");
}

VoidT F_ApiCommand(command)
   IntT command;
{
   F_ObjHandleT docId, pgfId;
   docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
   switch(command)
   {
   case SAVE:  /* Get and save the insertion point. */
      tr = F_ApiGetTextRange(FV_SessionId, docId,
                FP_TextSelection);
      if (!tr.beg.objId) return;

      /* Get UID of paragraph containing insertion point.*/
      pgfUID = F_ApiGetInt(docId, tr.beg.objId, FP_Unique);
      break;
```

```
    case LAST: /* Scroll to saved insertion point. */
        if (!pgfUID) break; /* Break if no insertion point. */

        /* Get paragraph ID from its UID. Then scroll to it. */
        pgfId = F_ApiGetUniqueObject(docId, FO_Pgf, pgfUID);
        tr.beg.objId = tr.end.objId = pgfId;
        F_ApiScrollToText(docId, &tr);
    }
}
```

See also

F_ApiGetNamedObject().

F_ApiGetUpdateBookDefaultParams()

`F_ApiGetUpdateBookDefaultParams()` gets a default property list that you can use to call `F_ApiUpdateBook()`.

Synopsis

```
#include "fapi.h"
. . .
F_PropValsT F_ApiGetUpdateBookDefaultParams();
```

Arguments

None.

Returns

A property list (an `F_PropValsT` data structure) with the properties shown in the following table. The first value listed by each property is the value that `F_ApiGetUpdateBookDefaultParams()` assigns to the property. The other values are values you can set the property to.

F_ApiGetUpdateBookDefault Params() property	Instruction or situation and possible values
`FS_AlertUserAboutFailure`	Alert user with warnings and messages if necessary.
	`False`: don't notify user when unexpected conditions occur.
	`True`: notify user when unexpected conditions occur.

F_ApiGetUpdateBookDefaultParams() property	Instruction or situation and possible values
FS_AllowInconsistentNumProps	Allow the FrameMaker product to update numbering, text insets, etc. of all the FrameMaker documents in the book, even if there are documents in the book with numbering properties that don't match the properties specified in the book.
	FV_DoOK: update numbering even if there are inconsistent properties in the book
	FV_DoCancel: cancel the update operation when it encounters a document with inconsistent numbering properties
	FV_DoShowDialog: show dialog box and let user decide
FS_AllowNonFMFiles	Allow the FrameMaker product to update numbering, text insets, etc. of all the FrameMaker documents in the book, even if there are documents in the book that were not created by a FrameMaker product.
	FV_DoOK: update the book even if the book contains files not created by a FrameMaker product
	FV_DoCancel: cancel the update operation when it encounters a document not created by a FrameMaker product
	FV_DoShowDialog: show dialog box and let user decide
FS_AllowViewOnlyFiles	Allow the FrameMaker product to update view-only documents in the book.
	FV_DoOK: update the view-only documents
	FV_DoCancel: cancel the entire update operation when it encounters a view-only document
	FV_DoShowDialog: show dialog box and let user decide
FS_MakeVisible	Make newly generated files (lists and indexes) visible.
	True: make visible
	False: don't make visible
FS_NumExportParams	The available number of properties in the export script that user can set.
FS_NumExportReturnParams	The number of parameters returned by the export script.

F_ApiGetUpdateBookDefaultParams() property	Instruction or situation and possible values
FS_ShowBookErrorLog	Display the book error log for this update operation.
	False: don't display the error log; all warnings and errors go to the console
	True: display the error log
FS_UpdateBookGeneratedFiles	Update generated files such as TOC, lists, and indexes. Only update those generated files that have FP_GenerateInclude set to True in their associated FO_BookComponent objects.
	True: update generated files
	False: don't update generated files
FS_UpdateBookNumbering	Update numbering in all the book's documents.
	True: update numbering
	False: don't update numbering
FS_UpdateBookOleLinks	Update OLE links in all the book's documents.
	True: update OLE links
	False: don't update OLE links
FS_UpdateBookTextReferences	Update text insets in all the book's documents.
	True: update text insets
	False: don't update text insets
FS_UpdateBookXRefs	Update cross-references in all the book's documents.
	True: update cross-references
	False: don't update cross-references

IMPORTANT: *The returned* `F_PropValsT` *structure references memory that is allocated by the API. Use* `F_ApiDeallocatePropVals()` *to free this memory when you are done with it.*

If `F_ApiGetOpenDefaultParams()` fails, the API sets the `len` field of the returned structure to `0`.

Example

The following code gets a default Update Book script with `F_ApiGetUpdateBookDefaultParams()`. It modifies the script to instruct the FrameMaker product to prompt the user if the book contains non-FrameMaker files. It then uses the script to call `F_ApiUpdateBook()`.

```
. . .
IntT i;
F_PropValsT script, *returnp = NULL;
F_ObjHandleT bookId;

bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook);
if(!bookId)
    return;

/* Get default property list. */
script = F_ApiGetUpdateBookDefaultParams();

/* Get indexes of properties and change them. */
i = F_ApiGetPropIndex(&script, FS_AlertUserAboutFailure);
script.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&script, FS_AllowNonFMFiles);
script.val[i].propVal.u.ival = FV_DoShowDialog;

/* Update the book */
F_ApiUpdateBook(bookId, &script, &returnp);

/* Free memory used by Update Book scripts. */
F_ApiDeallocatePropVals(&script);
F_ApiDeallocatePropVals(returnp);
. . .
```

See also

F_ApiUpdateBook().

F_ApiGetVal()

`F_ApiGetVal()` queries a property of any type. If you know a property's type, it is normally easier to get its value by calling an `F_ApiGet()` function, such as `F_ApiGetInt()`.

Synopsis

```
F_TypedValT F_ApiGetVal(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to query. If the object is a session, specify `0`.
`objId`	The ID of the object whose property you want to query.
`propNum`	The property to query. Specify an API-defined constant, such as `FP_FnNum`.

Returns

The `F_TypedValT` structure for the specified property.

If `F_ApiGetVal()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code prints the name of the current FrameMaker product:

```
. . .
F_TypedValT val;

val = F_ApiGetVal(FV_SessionId, FV_SessionId, FP_ProductName);
F_Printf(NULL,"The product name is %s.\n", val.u.sval);
. . .
```

See also

F_ApiDeallocateStructureType() and F_ApiGetProps().

F_ApiHypertextCommand()

`F_ApiHypertextCommand()` simulates a user-invoked hypertext command. Calling `F_ApiHypertextCommand()` has the same effect as a user clicking on a hypertext marker containing the specified text.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiHypertextCommand(F_ObjHandleT docId,
    StringT hypertext);
```

Arguments

`docId`	The ID of the document that you want to act as the source of the hypertext command.
`hypertext`	A hypertext command to execute, such as `gotolink` or `previouslink`. You can specify any command that would be valid in a hypertext marker in the document specified by `docId`.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiHypertextCommand()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID

Example

The following code executes a hypertext command to return to the previous hypertext link:

```
. . .
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
F_ApiHypertextCommand(docId, "previouslink");
. . .
```

F_ApiImport()

`F_ApiImport()` imports text or graphics into a document.

`F_ApiImport()` allows you to specify a script (property list) telling the FrameMaker product how to import text or graphics and how to deal with error and warning conditions. For example, you can specify whether to import a file by reference or by copy.

If you import a file by reference, `F_ApiImport()` creates an inset. The following table summarizes the types of files you can import with `F_ApiImport()` and the types of inset objects it creates when you import them by reference.

File type	Type of inset object F_ApiImport() creates
Graphics	`FO_Inset`
Text	`FO_TiText`
	`FO_TiTextTable`
Frame binary document	`FO_TiFlow`
MIF	`FO_TiFlow`

When importing a graphic, you can specify that it be imported at its default resolution by setting the `FS_GraphicDpi` property to 0 and setting the `FS_FitGraphicInSelectedRect` property to False. If the graphic has no default resolution, it is imported at 72 dpi.

For graphic objects, FrameMaker automatically determines whether the object-path is a local path or an HTTP path and acts accordingly.For example, if the path is an HTTP path, the object is downloaded before importing it in the file.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiImport(F_ObjHandleT enclosingDocId,
    F_TextLocT *textLocP,
    StringT filename,
    F_PropValsT *importParamsp,
    F_PropValsT **importReturnParamspp);
```

Arguments

`enclosingDocId`	The ID of the document into which to import the file.
`textLocP`	The text location at which to import the file.
`fileName`	The full pathname of the file to import. For information on how to specify pathnames on different platforms, see the *FDK Platform Guide* for your platform.
`importParamsp`	A property list telling the FrameMaker product how to import the file and how to respond to errors and other conditions. To use the default list, specify `NULL`.
`importReturnParamspp`	A property list that provides information about how the FrameMaker product imported the file. It must be initialized before you call `F_ApiImport()`.

IMPORTANT: *Always initialize the pointer to the property list that you specify for* `importReturnParamspp` *to* `NULL` *before you call* `F_ApiImport()`.

To get a property list to specify for the `importParamsp` parameter, use `F_ApiGetImportDefaultParams()` or create the list from scratch. For a list of all the properties an Import script property list can include, see "F_ApiGetImportDefaultParams()" on page 193.

Returns

The ID of the inset that `F_ApiImport()` creates. Or `0` if `F_ApiImport()` imports by copy or creates a graphic inset.

If `F_ApiImport()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_SystemError`	System error, such as an unreadable file or insufficient memory
`FE_BadParameter`	The property list contained an invalid parameter
`FE_BadFileType`	The specified file exists, but it does not have the correct file type
`FE_MissingFile`	The specified file does not exist
`FE_NoSuchFlow`	The script specifies an import flow that does not exist
`FE_FailedState`	Internal error

FA_errno value	Meaning
FE_CircularReference	Importing the specified file causes a circular reference
FE_FileClosedByClients	The file was closed by a client before it could be imported

The property list returned to importReturnParamspp has the properties shown in the following table.

Property	Meaning and Possible Values
FS_ImportedFileName	A string specifying the source file's pathname. If you scripted FS_ShowBrowser, this pathname can be different from the one you specified in the Import script.
FS_ImportNativeError	The error condition; normally the same value as FA_errno. If the file is imported successfully, it is set to FE_Success. See the table below for a list of possible values.
FS_ImportStatus	A bit field indicating what happened when the file was imported. See the table below for a list of possible flags.

Both the FS_ImportNativeError property and the FA_errno global variable indicate the result of a call to F_ApiImport(). The following table lists the possible

status flags and the `FA_errno` and `FS_ImportNativeError` values associated with them.

FS_ImportNativeError and FA_errno values	Possible FS_ImportStatus flags
`FE_BadParameter`, `FE_BadFileType`, `FE_MissingFile`, `FE_FailedState`, or `FE_CanceledByClient` (file was not imported)	`FV_BadImportFileName`: the specified source filename is invalid
	`FV_BadImportFileType`: the Import script specified a file type different from the source file's actual type
	`FV_BadImportScriptValue`: the Import script contained an invalid property value
	FV_BadTextFileTypeHint: The file was a text file, and the string in FS_FileTypeHint was not a valid import hint string. For information on the syntax of this string, see "Syntax of FP_ImportHint strings" on page 939.
	`FV_MissingScript`: `F_ApiImport()` was called without a script.
	`FV_DisallowedImportType`: source file's type disallowed by script.
	`FV_NoMainFlow`: script specified to import the main flow, but the source file does not have a main flow.
	`FV_NoFlowWithSpecifiedName`: script specified a flow name that does not exist
	`FV_InsertionPointNotInText`: the insertion point in the enclosing document is not in text
	`FV_InsufficientMemory`: there is insufficient memory to import the source file
	`FV_BadEnclosingDocId`: there is no open document with the specified ID
	`FV_ImportFileNotReadable`: the specified source file is unreadable

FS_ImportNativeError and FA_errno values	Possible FS_ImportStatus flags
FE_Success	FV_ImportedByCopy: the source file was imported by copy
	FV_ImportedText: the source file is a text file
	FV_ImportTextTable: the source file is a text file, which was imported into a table.
	FV_ImportedMIF: the source file is a MIF file
	FV_ImportedMakerDoc: the source file is a FASL file.
	FV_ImportedFilteredFile: the source file was filtered
	FV_ImportedGraphicFile: the source file is a graphics file
	FV_ImportedSgmlDoc: the source file is an SGML document
	FV_ImportedXmlDoc: the source file is an XML document

FS_ImportNativeError and FA_errno values	Possible FS_ImportStatus flags
FE_Cancel	FV_CancelFileText: the file is text, so the user or the Import script canceled the Import operation
	FV_CancelFileGraphic: the source file is a graphic, so the user or the Import script canceled the Import operation
	FV_CancelFileMacEdition: the source file is a Macintosh Edition, so the Import script canceled the Import operation
	FV_CancelFileDoc: the file is a FASL file, so the user or the script canceled the Import operation
	FV_CancelFileSgml: the file is an SGML document, so the user or the script canceled the Import operation
	FV_CancelFileXML: the file is an XML document, so the user or the script canceled the Import operation
	FV_CancelFileMIF: the source file is a MIF file, so the user or the script canceled the Import operation
	FV_CancelFileFilterable: the source file is a filterable file, so the user or the script canceled the Import operation
	FV_InsertionPointInFootnote: the insertion point was in a footnote and the import script specified to import the file as a table, so the file could not be imported
	FV_InsertionPointInTableCell: the insertion point was in a table cell and the import script specified to import the file as a table, so the file could not be imported
	FV_UserCanceledImport: the user canceled the Import operation
	FV_UserCanceledImportBrowser: the user canceled the Import browser

To determine whether a particular FS_ImportStatus bit is set, use F_ApiCheckStatus(). For more information, see "F_ApiCheckStatus()" on page 96.

After you are done with the property lists you use to call F_ApiImport(), use F_ApiDeallocatePropVals() to deallocate the memory they use. Calling F_ApiImport() again also deallocates the memory.

Example

The following code imports a graphic file, named `agraphic.xwd`, by reference.

```
. . .
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT docId;
F_TextRangeT tr;
IntT i;

/* Get default import list. Return if it can't be allocated. */
params = F_ApiGetImportDefaultParams();
if(params.len == 0) return;

/* Get current insertion point. Return if there isn't one. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(0, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Change properties to disallow nongraphic files. */
i = F_ApiGetPropIndex(&params, FS_DisallowDoc);
params.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&params, FS_DisallowMIF);
params.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&params, FS_DisallowPlainText);
params.val[i].propVal.u.ival = True;


/* Set properties to import graphic at default resolution*/
i = F_ApiGetPropIndex(&params, FS_GraphicDpi);
params.val[i].propVal.u.ival = 0;
i = F_ApiGetPropIndex(&params, FS_FitGraphicInSelectedRect);
params.val[i].propVal.u.ival = False;

F_ApiImport(docId, &tr.beg, "/tmp/agraphic.xwd",
             &params, &returnParamsp);

if (F_ApiCheckStatus(returnParamsp, FV_BadImportFileType))
   F_ApiAlert("File isn't a graphic.", FF_ALERT_CONTINUE_NOTE);

/* Deallocate property lists. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
. . .
```

See also

"F_ApiCheckStatus()" on page 96, "F_ApiGetOpenDefaultParams()" on page 219, "F_ApiPrintOpenStatus()" on page 370, and "F_ApiSimpleOpen()" on page 465.

F_ApiInitialize()

`F_ApiInitialize()` is a callback that you define in your client to respond to a FrameMaker product attempting to initialize your client.

`F_ApiInitialize` should be called for all APIs clients, including filters and document reports. This enables filters and document reports to make the `F_ApiEnableUnicode(True)` call necessary for enabling Unicode Mode. If Unicode Mode isn't enabled, sparm received in `F_ApiNotify` isn't in Unicode because the API runs in Compatibility Mode. For more information on Unicode Mode and Compatibility Mode, see Chapter 1, "Working with Unicode." in the *FDK Programmer's Reference*.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiInitialize(initialization)
    IntT initialization;
```

Arguments

`initialization`	A constant that indicates the type of initialization

The following table summarizes the different types of initializations and the `initialization` constants FrameMaker products can use to call call `F_ApiInitialize()`.

Type of initialization	When F_ApiInitialize() is called	Initialization flag	Clients that receive initialization
FrameMaker product starts with no special options	After starting	`FA_Init_First`	All except document reports and filters
FrameMaker product starts with take-control client	After starting	`FA_Init_First`	All except document reports and filters
	After all API clients have finished processing the `FA_Init_First` initialization	`FA_Init_TakeControl`	Client that takes control of the session
Document report chosen from Document Reports dialog box	After report is chosen	`FA_Init_DocReport`	The chosen document report
Notification, menu choice, or hypertext command for a client that has bailed out	When the menu item is chosen, the hypertext command is clicked, or the notification should be issued	`FA_Init_Subsequent`	Clients that have bailed out and are waiting for an event, a menu choice, or a hypertext command to occur

Returns

`VoidT`.

Example

The following code displays a dialog box after the FrameMaker product is started:

```
. . .
VoidT F_ApiInitialize(initialization)
IntT initialization;
{
if (initialization == FA_Init_First)
 F_ApiAlert("API client has started.", FF_ALERT_CONTINUE_NOTE);
}
. . .
```

See also

"F_ApiNotification()" on page 339.

F_ApiIsEncodingSupported()

`F_ApiIsEncodingSupported()` checks whether the specified encoding is supported for the current session. For example, unless FrameMaker is running on a system that supports Japanese text, Japanese encoding is not supported.

Synopsis

```
#include "fapi.h"
. . .
BoolT F_ApiIsEncodingSupported(ConStringT encodingName);
```

Arguments

`encodingName` The encoding you want to test.

Allowed values for encodingName are:

Value	Means
`FrameRoman`	Roman text
`JISX0208.ShiftJIS`	Japanese text
`BIG5`	Traditional Chinese text
`GB2312-80.EUC`	Simplified Chinese text
`KSC5601-1992`	Korean text

Returns

`True` if the specified encoding is supported for the current session; otherwise returns `False`.

Example

The following code checks to see whether Japanese text is supported for the current session:

```
. . .
BoolT isSupported;

isSupported = F_ApiIsEncodingSupported(
                              (ConString) "JISX0208.ShiftJIS");
. . .
```

F_ApiLoadMenuCustomizationFile()

`F_ApiLoadMenuCustomizationFile()` loads a menu customization file. A menu customization file is a text file containing statements that change the menus and commands the user sees in the FrameMaker product. For example, a menu customization file can change the name of a command or move a command from one menu to another.

For information on writing menu customization files, see the online manual *Customizing FrameMaker products*.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiLoadMenuCustomizationFile(StringT pathname,
    BoolT silent);
```

Arguments

`pathname`	The pathname of the menu customization file to load. If you specify only a filename, the function looks in the client directory. If `silent` is set to `False`, the pathname specified by `pathname` is used as the default in the Menu Customization File dialog box.
`silent`	Specifies whether to display the Menu Customization File dialog box and allow the user to choose the file. To display the dialog box and allow the user to choose the file, specify `False`. To use the file specified by `pathname` without asking the user, specify `True`.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiLoadMenuCustomizationFile()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_BadOperation`	Parameters specify an invalid operation
`FE_BadParameter`	Parameter has an invalid value
`FE_SystemError`	System error

Example

The following code loads a menu customization file named `mymenus.cfg`, without notifying the user:

```
. . .
F_ApiLoadMenuCustomizationFile("c:\\maker\\fmint\\mymenus.cfg",
             True);
. . .
```

F_ApiMakeTblSelection()

`F_ApiMakeTblSelection()` selects a range of cells in a table.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiMakeTblSelection(F_ObjHandleT docId,
   F_ObjHandleT tableId,
   IntT topRow,
   IntT bottomRow,
   IntT leftCol,
   IntT rightCol);
```

Arguments

`docId`	The ID of the document containing the table.
`tableId`	The ID of the table containing the rows to select.
`topRow`	The number of the first row in the selection. The rows are numbered from top to bottom, starting with 0 (including heading rows). To select the entire table, specify `FF_SELECT_WHOLE_TABLE`.
`bottomRow`	The number of the last row in the selection.
`leftCol`	The number of the leftmost column in the selection. The columns are numbered from left to right, starting with 0.
`rightCol`	The number of the rightmost column in the selection.

To select an entire table, including the table title, set `topRow` to `FF_SELECT_WHOLE_TABLE`. `F_ApiMakeTblSelection()` ignores the values for the other parameters.

IMPORTANT: `F_ApiMakeTblSelection()` *can't select different types of rows at the same time, unless you set* `topRow` *to* `FF_SELECT_WHOLE_TABLE` *or you set* `topRow` *and* `bottomRow` *to select one or more entire columns.*

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiMakeTblSelection()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid table ID
`FE_BadOperation`	Parameters specify an action that is invalid
`FE_OutOfRange`	Row or column specification not in table
`FE_BadParameter`	Parameter has an invalid value

Example

The following code selects the cells R1-C1 to R2-C2 in a table:

```
. . .
F_ObjHandleT docId, tblId;

/* Get the document and table IDs. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tblId = F_ApiGetId(FV_SessionId, docId, FP_FirstTblInDoc);

/* Select the cells. If row 0 is a heading row and
** row 1 is a body row, this call won't work.
*/
F_ApiMakeTblSelection(docId, tblId, 0, 1, 0, 1);
. . .
```

F_ApiMenuItemInMenu()

`F_ApiMenuItemInMenu()` determines if a menu item or menu is on a menu or menu bar.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiMenuItemInMenu (F_ObjHandleT menuId,
    F_ObjHandleT menuitemId,
    BoolT recursive);
```

Arguments

`menuId`	The menu or menu bar to search.
`menuitemId`	The ID of the menu item or menu to search for.
`recursive`	Specifies whether to search the submenus of the menu or menu bar specified by `menuId`. Specify `True` to search them.

Returns

The ID of the menu on which `F_ApiMenuItemInMenu()` found the object specified by `menuitemId`, or `0` if it did not find the object.

If `F_ApiMenuItemInMenu()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation or `fmbatch` is running
`FE_BadOperation`	Parameters specify an invalid operation
`FE_BadParameter`	Parameter has an invalid value

Example

The following code searches the Edit menu and its submenus for the Copy menu item:

```
. . .
F_ObjHandleT copyCmdId, menuId, returnId;

menuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                    "EditMenu");
copyCmdId = F_ApiGetNamedObject(FV_SessionId, FO_Command,
                    "Copy");
returnId = F_ApiMenuItemInMenu(menuId, copyCmdId, True);

if(returnId == menuId)
   F_ApiAlert("Copy is on the Edit menu." ,
                    FF_ALERT_CONTINUE_NOTE);

else if (!returnId)
   F_ApiAlert("Copy is not on the Edit menu." ,
                    FF_ALERT_CONTINUE_NOTE);
else
   F_ApiAlert("Copy is on a pull-right of the Edit menu.",
                    FF_ALERT_CONTINUE_NOTE);
. . .
```

See also

"F_ApiGetNamedObject()" on page 214.

Structured F_ApiMergeIntoFirst()

`F_ApiMergeIntoFirst()` merges the selected structural elements into the first element in the selection.

IMPORTANT: *At least two structural elements must be selected when* `F_ApiMergeIntoFirst()` *is called.*

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiMergeIntoFirst(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document containing the selected elements

Returns

`VoidT`.

If `F_ApiMergeIntoFirst()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product isn't supported
`FE_BadDocId`	Invalid document ID
`FE_BadSelectionForOperation`	Current text selection is invalid for this operation

Example

The following code merges the selected structural elements in the active document:

```
. . .
F_ObjHandleT docId;

/* Get the document ID. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Merge the elements. */
F_ApiMergeIntoFirst(docId);

. . .
```

See also

"F_ApiMergeIntoLast()".

Structured **F_ApiMergeIntoLast()**

`F_ApiMergeIntoLast()` merges the selected structural elements into the last element in the selection.

IMPORTANT: *At least two structural elements must be selected when* `F_ApiMergeIntoLast()` *is called.*

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiMergeIntoLast(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document containing the selected elements

Returns

`VoidT`.

If `F_ApiMergeIntoLast()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product isn't supported
`FE_BadDocId`	Invalid document ID
`FE_BadSelectionForOperation`	Current text selection is invalid for this operation

Example

The following code merges the selected structural elements in the active document:

```
. . .
F_ObjHandleT docId;

/* Get the document ID. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Merge the elements. */
F_ApiMergeIntoLast(docId);
. . .
```

See also

"F_ApiMergeIntoFirst()" on page 300.

F_ApiMessage()

`F_ApiMessage()` is a callback that you can define in your client. The FrameMaker product calls `F_ApiMessage()` when the user clicks a hypertext marker containing the text `message` *`apiclient`*, where *`apiclient`* is the name under which your client is registered with the FrameMaker product. For information on creating hypertext markers that message FDK clients, see "Using hypertext commands in your client's user interface" on page 213 of the *FDK Programmer's Guide*.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiMessage(StringT message,
    F_ObjHandleT docId,
    F_ObjHandleT objId);
```

Arguments

`message`	The string specified by the hypertext command. It is `NULL` if the user clicked an inset.
`docId`	The ID of the document containing the hypertext marker or the inset.
`objId`	The ID of the hypertext marker or the inset. If the user chooses your client from the Inset Editors scroll list, it is `0`.

Returns

`VoidT`.

Example

See "Responding to message apiclient commands" on page 214 and "Responding to the user launching your inset editor" on page 494 of the *FDK Programmer's Guide*.

See also

"F_ApiHypertextCommand()" on page 285 and "F_ApiNotify()" on page 349.

F_ApiModalDialog()

`F_ApiModalDialog()` displays a dialog resource as a modal dialog box. This is the most common type of dialog box. The user must close the dialog box before continuing, usually by clicking OK or Cancel.

IMPORTANT: *When you are through with it, you should always close a dialog resource via* `F_ApiClose()`. *Also, in some circumstances you might want a custom dialog box to be modal, and in others you might want it to be modeless. To do this, you should create two separate dialog resources. Then you should always call one with* `F_ApiModalDialog()`, *and call the other with* `F_ApiModelessDialog()`.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiModalDialog(IntT dlgNum,
   F_ObjHandleT dlgId);
```

Arguments

`dlgNum`	A unique number to identify the dialog box. The API passes this number to your client's `F_ApiDialogEvent()` callback when there is a user action in the dialog box. If you don't want the API to call your client's `F_ApiDialogEvent()` callback when there is a user action, set `dlgNum` to `0`.
`dlgId`	The ID of the dialog resource to display.

After calling `F_ApiModalDialog()`, you must call `F_ApiClose()` to free memory.

Returns

`FE_Success` or a nonzero value if the user requested help.

Example

The following code displays a dialog named mydlg as a modal dialog box:

```
. . .
#define MY_DLG 1
F_ObjHandleT dlgId;

/* Open the resource and display the dialog box. */
dlgId =
Resource(FO_DialogResource, "mydlg");

F_ApiModalDialog(MY_DLG, dlgId);
. . .
F_ApiClose (dlgId, FF_CLOSE_MODIFIED);
. . .
```

See also

"F_ApiModelessDialog()" on page 305, "F_ApiDialogEvent()" on page 155, and "F_ApiOpenResource()" on page 363.

F_ApiModelessDialog()

`F_ApiModelessDialog()` displays a dialog resource as a modeless dialog box. Modeless dialog boxes allow the user to switch between the dialog box and the window that created it. Modeless dialog boxes are preferred when it would be convenient to keep the dialog box displayed for a while.

IMPORTANT: *When you are through with it, you should always close a dialog resource via* `F_ApiClose()`*. Also, in some circumstances you might want a custom dialog box to be modal, and in others you might want it to be modeless. To do this, you should create two separate dialog resources. Then you should always call one with* `F_ApiModalDialog()`*, and call the other with* `F_ApiModelessDialog()`*.*

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiModelessDialog(IntT dlgNum,
    F_ObjHandleT dlgId);
```

Arguments

`dlgNum`	A unique number to identify the dialog box. The API passes this number to your client's `F_ApiDialogEvent()` callback when there is a user action in the dialog box.
`dlgId`	The ID of the dialog resource to display.

After calling `F_ApiModelessDialog()`, you must call `F_ApiClose()` to free memory.

Returns

`FE_Success` or an error code if the dialog resource couldn't be displayed.

Example

The following code displays a dialog named mydlg as a modeless dialog box:

```
. . .
#define MY_DLG 1
F_ObjHandleT dlgId;

/* Open the resource and display the dialog box. */
dlgId = F_ApiOpenResource(FO_DialogResource, "mydlg");

F_ApiModelessDialog(MY_DLG, dlgId);
. . .
F_ApiClose (dlgId, FF_CLOSE_MODIFIED);
. . .
```

See also

"F_ApiModalDialog()" on page 303, "F_ApiDialogEvent()" on page 155, and "F_ApiOpenResource()" on page 363.

F_ApiMoveComponent()

Moves a book component within the book.

Synopsis

```
#include "fapi.h"
...
VoidT F_ApiMoveComponent(F_ObjHandleT bookId,
    F_ObjHandleT compId,
    IntT moveAction);
```

Arguments

`bookId` .	The ID of the book that contains the component
`compId`	The ID of the book component that is to be moved.
`moveAction` .	Specifies the action to move the component

You can specify the following values for `moveAction`.

moveAction	Meaning
FA_COMPONENT_MOVEUP	Move the component up at the same hierarchical level.
FA_COMPONENT_MOVEDOWN	Move the component down at the same hierarchical level.
FA_COMPONENT_PROMOTE	Move the component to a higher/outer level in hierarchy.
FA_COMPONENT_DEMOTE	Move the component to a lower/inner level in hierarchy.

Returns

VoidT

Example

The following code moves up a book component.

```
. . .
F_ObjHandleT bookId;
F_ObjHandleT comp1Id, comp2Id;

/* Get ID of active book and its second component. */
bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook);
comp1Id = F_ApiGetId(FV_SessionId, bookId,
          FP_FirstComponentInBook);
comp2Id = F_ApiGetId(bookId, comp1Id,
          FP_NextComponentInBook);
/* Move the component. */
F_ApiMoveComponent(bookId, comp2Id,
          FA_COMPONENT_MOVEUP);
...
```

F_ApiNetLibSetAuthFunction()

Sets a callback function that is called for setting the username/password information before performing the NetLib related authentication.

Synopsis

```
#include "fapi.h"
...
VoidT F_ApiNetLibSetAuthFunction(VoidT (*p_auth_func)(ConStringT
url,
StringT username, StringT password,
IntT *cancelp) );
```

Arguments

`p_auth_func`	Callback function that sets the username and password information for the NetLib related authentication.

:

NOTE: After registering the callback function, whenever NetLib needs authentication, it calls this function with the server's URL as '`url`' parameter. This function checks the URL string and accordingly fills-in the username and password fields and sets *cancelp to 0. If you want to cancel the authentication request from NetLib, set *cancelp to 1. In this case, set the username and password to empty strings.

To unregister the callback function, call `F_ApiNetLibSetAuthFunction`() with NULL as the parameter. Then, FrameMaker's standard NetLib authentication dialog is displayed for the purpose.

The server authentication fails if the strings are set with an invalid username or password through this callback function. NetLib calls the function again and this time too, the function sets the wrong authentication information. This process will go on and FrameMaker will hang. Therefore, use some mechanism (like counters) in the callback function to avoid such a condition. For example, if the authentication requests for a particular server reaches a certain number of times (say 3), cancel the later requests for that server.

Returns

`VoidT`

Example

The following code sets the authentication callback function as 'SplAuthFunction()'. The callback function matches the URL provided and fills in the username & password

accordingly.

```
. . .
   F_ApiNetLibSetAuthFunction(SplAuthFunction);
. . .
. . .
VoidT SplAuthFunction(ConStringT url, StringT username,
                      StringT password, IntT *cancelp)
{
       static int attempts_srv1 = 0;
       static int attempts_srv2 = 0;

       // send authentication information for first three
       attempts only
   if (attempts_srv1 < 3 &&
             F_StrCmp(url, (ConStringT)"hostname:8080") == 0)
   {

          F_StrCpy(username, (ConStringT)"user123");
          F_StrCpy(password, (ConStringT)"pass234");
          *cancelp = 0;
          attempts_srv1++;
   }
   else if (attempts_srv2 < 3 &&
          F_StrCmp(url, (ConStringT)"cmsserver") == 0)
   {
          F_StrCpy(username, (ConStringT)"Admin");
          F_StrCpy(password, (ConStringT)"Password");
          *cancelp = 0;
          attempts_srv2++;
   }
   else
   {
          F_StrCpy(username, (ConStringT)"");
          F_StrCpy(password, (ConStringT)"");
          *cancelp = 1;
          attempts_srv1 = 0;
```

```
            attempts_srv2 = 0;
    }
}
. . .
```

F_ApiNewAnchoredFormattedObject()

`F_ApiNewAnchoredFormattedObject()` can create the following types of anchored objects:

- `FO_Var`
- `FO_XRef`
- `FO_Tbl`

`F_ApiNewAnchoredFormattedObject()` inserts the object at the specified location in text. It uses arbitrary default properties for the new object. After you have created the object, you can use the `F_ApiSet()` functions to change its properties.

If you call `F_ApiNewAnchoredFormattedObject()` to create a table, it uses the default numbers of rows and columns from the specified Table Catalog format. To use the default Table Catalog format for a new table, set `format` to `NULL`. To specify the number of rows and columns when you create a table, use `F_ApiNewTable()`.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewAnchoredFormattedObject(F_ObjHandleT docId,
    IntT objType,
    StringT format,
    F_TextLocT *textLocp);
```

Arguments

`docId`	The ID of the document to which to add the object
`objType`	The type of object to create (for example, `FO_XRef`)
`format`	The string that specifies the object's format (for example, `Heading & Page` for a cross-reference, `Format A` for a table, or `Current Date (Long)` for a variable)
`textLocp`	The text location at which to insert the anchored object

The `F_TextLocT` structure is defined as:

```
typedef struct {
   F_ObjHandleT objId; /* ID of the paragraph or text line */
   IntT offset; /* From beginning of paragraph or text line */
} F_TextLocT;
```

Returns

The ID of the new anchored object, or `0` if an error occurs.

If `F_ApiNewAnchoredFormattedObject()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_NotTextObject`	Object specified for text location is not a paragraph (`FO_Pgf`)
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified paragraph or text line
`FE_BadNew`	Object can't be created; the format specified by `format` may not exist

Example

The following code adds a `Current Date (Short)` variable at the insertion point (or the beginning of the text selection) of the active document:

```
. . .
F_TextRangeT tr;
F_ObjHandleT docId, variableId;

/* Get the insertion point. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (tr.beg.objId == 0) return;

/* Insert the variable. */
variableId = F_ApiNewAnchoredFormattedObject(docId, FO_Var,
             "Current Date (Short)", &tr.beg);
. . .
```

See also

"F_ApiNewAnchoredObject()".

F_ApiNewAnchoredObject()

`F_ApiNewAnchoredObject()` can create any of the following anchored objects:

- `FO_AFrame`
- `FO_Fn`
- `FO_Marker`
- `FO_TiApiClient`
- `FO_Tbl`

`F_ApiNewAnchoredObject()` inserts the object at the specified location in text. It uses arbitrary default properties for the new object. After you have created the object, you can use the `F_ApiSet()` functions to change its properties.

Tables created by `F_ApiNewAnchoredObject()` have a single column and a single body row. It is usually easier to use `F_ApiNewTable()` to create tables.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewAnchoredObject(F_ObjHandleT docId,
    IntT objType,
    F_TextLocT *textlocp);
```

Arguments

`docId`	The ID of the document to which to add the object
`objType`	The type of object to create (for example, `FO_Marker` or `FO_Fn`)
`textlocp`	The text location at which to insert the anchored object

The `F_TextLocT` structure is defined as:

```
typedef struct {
    F_ObjHandleT objId; /* ID of the paragraph or text line */
    IntT offset; /* From beginning of paragraph or text line */
} F_TextLocT;
```

Returns

The ID of the new anchored object, or `0` if an error occurs.

If `F_ApiNewAnchoredObject()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadNew`	Anchored object can't be created
`FE_BadObjId`	Invalid object ID
`FE_BadOperation`	Function call specified an illegal operation
`FE_NotTextObject`	Object specified for the text location is not a paragraph (`FO_Pgf`)
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified paragraph or text line

Example

The following code inserts a new anchored frame at the beginning of a paragraph:

```
. . .
F_TextRangeT tr;
F_ObjHandleT docId, afrmId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Insert the frame. */
afrmId = F_ApiNewAnchoredObject(docId, FO_AFrame, &tr.beg);
. . .
```

See also

"F_ApiNewAnchoredFormattedObject()" on page 310.

Structured F_ApiNewBookComponentInHierarchy()

`F_ApiNewBookComponentInHierarchy()` inserts a book component at a specified position in a structured book.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewBookComponentInHierarchy(
    F_ObjHandleT bookId;
    StringT compName,
    F_ElementLocT *elemLocp);
```

Arguments

`bookId`	The ID of the book to add the component to
`compName`	The name of the component
`elemLocp`	The position at which to add the new element

IMPORTANT: *The book you specify for* `bookId` *must already be structured.*

Returns

The ID of the new element that corresponds to the book component, or `0` if an error occurs.

If `F_ApiNewBookComponentInHierarchy()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadBookId`	Invalid book ID
`FE_BadCompPath`	Component name specified for `compName` is invalid
`FE_BadNew`	The object can't be created
`FE_BookUnStructured`	Specified book is unstructured

Example

The following code adds a component to a book:

```
. . .
F_ObjHandleT bookId, elemId;
F_ElementLocT elemLoc;

/* Get ID of active book and its highest level element. */
bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook);
elemLoc.childId = F_ApiGetId(FV_SessionId, bookId,
                    FP_HighestLevelElement);
elemLoc.parentId = 0;
elemLoc.offset = 0;

/* Insert the new element. */
elemId = F_ApiNewBookComponentInHierarchy(bookId, "Chapter1",
                    &elemLoc);
. . .
```

See also

"F_ApiNewElementInHierarchy()" on page 320 and "F_ApiNewSeriesObject()" on page 329.

F_ApiNewBookComponentOfTypeInHierarchy()

Inserts a book component of a specified type at a specified position in a structured FrameMaker book.

Synopsis

```
#include "fapi.h"
...
F_ObjHandleT F_ApiNewBookComponentOfTypeInHierarchy(F_ObjHandleT
bookId,
ConStringT compName,
IntT compType,
const F_ElementLocT *elemLocp);
```

Arguments

You can specify the following values for `compType`.

compType	Meaning
FV_BK_FOLDER	Folder type book component.
FV_BK_GROUP	Group type book component.

Returns

The ID of the new element that corresponds to the book component, or 0 if an error occurs.
If `F_ApiNewBookComponentOfTypeInHierarchy()` fails, the API assigns one of the following values to `FA_errno`

FA_errno Value	Meaning
FE_BadBookId	Invalid book ID.
FE_BadParameter	Invalid compType.
FE_BadNew	The object can't be created.
FE_BookUnStructured	Specified book is unstructured.

Example

The following code adds a component of type folder to a book.

```
. . .
F_ObjHandleT bookId, elemId;
F_ElementLocT elemLoc;

/* Get ID of active book and its highest level element. */
bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook);
elemLoc.childId = F_ApiGetId(FV_SessionId, bookId,
FP_HighestLevelElement);

elemLoc.parentId = 0;
elemLoc.offset = 0;

/* Insert the new element. */
elemId = F_ApiNewBookComponentOfTypeInHierarchy(bookId,
                            "TechDoc Folder", FV_BK_FOLDER,
                            &elemLoc);
. . .
```

Structured F_ApiNewElement()

`F_ApiNewElement()` creates a structural element (`FO_Element`) in a structured document.

`F_ApiNewElement()` inserts the new element at the specified location in text, using the specified element definition.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewElement(F_ObjHandleT docId,
   F_ObjHandleT elemDefId,
   F_TextLocT *textLocp);
```

Arguments

`docId`	The ID of the document to which to add the element
`elemDefId`	The ID of the element definition for the new element
`textLocp`	The text location at which to insert the new element

The `F_TextLocT` structure is defined as:

```
typedef struct {
   F_ObjHandleT objId; /* ID of the paragraph or text line */
   IntT offset; /* From beginning of paragraph or text line */
} F_TextLocT;
```

For object (noncontainer) elements, `F_ApiNewElement()` inserts the appropriate type of object for the element. If there is a matching format rule, `F_ApiNewElement()` uses it to format the object. Otherwise, it uses one of the following default formats:

Object type	Object inserted	Format used if no matching format rule exists
`FV_FO_XREF`	Cross-reference	`Undefined XRef`
`FV_FO_EQN`	Equation	`medium`
`FV_FO_MARKER`	Marker	`Type 11`
`FV_FO_TBL`	Table with the format and number of rows and columns specified by the table format	Format A if it exists; otherwise, a table with a heading row, 8 body rows, a footing row, and 5 columns
`FV_FO_SYS_VAR`	Variable	`Filename (Long)`
`FV_FO_GRAPHIC`	A centered 1.0-inch by 1.0-inch anchored frame below the current position; cropped is off, and floating is on	

Returns

The ID of the new element, or `0` if an error occurs.

If `F_ApiNewElement()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadNew`	Object can't be created
`FE_BadObjId`	Invalid object ID
`FE_NotTextObject`	Object specified for the text location isn't a paragraph (`FO_Pgf`) or a text line (`FO_TextLine`)
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified paragraph or text line

Example

See "Creating objects" on page 357 of the *FDK Programmer's Guide*.

See also

"F_ApiUnWrapElement()" on page 479.

Structured F_ApiNewElementInHierarchy()

`F_ApiNewElementInHierarchy()` creates a structural element (`FO_Element` object) at a specified location in the element hierarchy of a structured document or book.

To create the root element for a book, you must use `F_ApiNewElementInHierarchy()`; you can't use `F_ApiNewElement()`. However, you can't use `F_ApiNewElementInHierarchy()` to add elements to an unstructured document. You must structure the document first by adding a root element with `F_ApiNewElement()`.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewElementInHierarchy(F_ObjHandleT docId,
    F_ObjHandleT elemDefId,
    F_ElementLocT *elemLocp);
```

Arguments

`docId`	The ID of the document or book to add the element to
`elemDefId`	The ID of the element definition for the new element
`elemLocp`	The location at which the element is inserted

For object (noncontainer) elements, `F_ApiNewElementInHierarchy()` inserts the appropriate type of object for the element. If there is a matching format rule, `F_ApiNewElementInHierarchy()` uses it to format the object. Otherwise, it uses one of the following default formats:

Object type	Object inserted	Format used if no matching format rule exists
`FV_FO_XREF`	Cross-reference	`Undefined XRef`
`FV_FO_EQN`	Equation	`medium`
`FV_FO_MARKER`	Marker	`Type 11`

Object type	Object inserted	Format used if no matching format rule exists
FV_FO_TBL	Table with the format and number of rows and columns specified by the table format	Format A if it exists; otherwise, a table with a heading row, 8 body rows, a footing row, and 5 columns
FV_FO_SYS_VAR	Variable	Filename (Long)
FV_FO_GRAPHIC	A centered 1.0-inch by 1.0-inch anchored frame below the current position	

Returns

The ID of the new element, or 0 if an error occurs.

If F_ApiNewElementInHierarchy() fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
FE_BadBookId	Invalid book (FO_Book object) ID
FE_BadObjId	Invalid element definition (FO_ElementDef object) ID
FE_BadInsertPos	elemLocP specifies an invalid place to insert the element; for example, it specifies a position before the highest element in the flow

Example

The following code adds a Para element at the insertion point, or the beginning of the current element selection:

```
. . .
F_ElementRangeT elemSelect;
F_ObjHandleT docId, elemId, paraEdefId;

/* Get ID of active document and the Para element definition. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
paraEdefId = F_ApiGetNamedObject(docId, FO_ElementDef, "Para");

/* Get current element selection in active document. */
elemSelect = F_ApiGetElementRange(FV_SessionId, docId,
                FP_ElementSelection);

if(elemSelect.beg.parentId == 0 || paraEdefId == 0)return;

/* Insert the new element. */
elemId = F_ApiNewElementInHierarchy(docId, paraEdefId,
             &elemSelect.beg);
. . .
```

F_ApiNewGraphicObject()

`F_ApiNewGraphicObject()` creates the following types of graphic objects:

- `FO_Arc`
- `FO_Ellipse`
- `FO_Flow`[1]
- `FO_Group`
- `FO_Inset`
- `FO_Line`
- `FO_Math`
- `FO_Polyline`
- `FO_Polygon`

1. To create a flow, you must create a text frame. See “Creating flows” on page 366 of the *FDK Programmer’s Guide*.

- FO_Rectangle
- FO_RoundRect
- FO_TextFrame
- FO_TextLine
- FO_UnanchoredFrame

To create an anchored frame, use F_ApiNewAnchoredObject().

If there is more than one object within the parent frame, F_ApiNewGraphicObject() adds the new API graphic object to the end of the linked list of child objects. That is, it puts it in the front of the back-to-front draw order. It automatically takes care of updating the object's FP_PrevGraphicInFrame and FP_NextGraphicInFrame properties. F_ApiNewGraphicObject() gives the new API graphic object a set of arbitrary default properties. For example, FP_LocX and FP_LocY are both 0. After you have created the object, you can use the F_ApiSet() functions to change its properties.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewGraphicObject(F_ObjHandleT docId,
    IntT objType,
    F_ObjHandleT parentId);
```

Arguments

docId	The ID of the document in which to create the new object
objType	The type of API graphic object to create (for example, FO_Rectangle or FO_Line)
parentId	The ID of the parent frame in which to create the object

Returns

The ID of the new graphic object, or 0 if an error occurs.

If F_ApiNewGraphicObject() fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
FE_BadDocId	Invalid document ID
FE_BadObjId	Invalid object ID

FA_errno value	Meaning
FE_NotFrame	Specified parent object is not a frame
FE_BadNew	Object can't be created

Example

The following code creates an ellipse and a text frame within the selected frame:

```
. . .
F_ObjHandleT docId, parentId, ellpsId, tFramId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
parentId = F_ApiGetId(FV_SessionId,
                      docId, FP_FirstSelectedGraphicInDoc);
if ((F_ApiGetObjectType(docId, parentId) != FO_AFrame) &&
    (F_ApiGetObjectType(docId, parentId) != FO_UnanchoredFrame))
    {
    F_ApiAlert("Select a frame first.", FF_ALERT_CONTINUE_WARN);
    return;
    }
ellpsId = F_ApiNewGraphicObject(docId, FO_Ellipse, parentId);
tFramId = F_ApiNewGraphicObject(docId, FO_TextFrame, parentId);
. . .
```

For an example of how to create a polygon, see "F_ApiSetPoints()" on page 429.

See also

"F_ApiNewAnchoredObject()" on page 312, "F_ApiNewNamedObject()", and "F_ApiNewSeriesObject()" on page 329.

F_ApiNewKeyCatalog()

Creates a new key catalog with the specified tag.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewKeyCatalog(StringT tag)
```

Arguments

`tag`	The tag of the new key catalog.

Returns

Returns the handle of the new key catalog.

If `F_ApiNewKeyCatalog()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadName`	The tag provided is not valid.
`FE_DupName`	A key catalog for the tag provided already exists.

F_ApiNewKeyDefinition()

Adds a new key definition to the specified key catalog.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiNewKeyDefinition(F_ObjHandleT keyCatalogId, StringT
key, StringT href, IntT srcType, StringT srcFile, IntT flags)
```

Arguments

`keyCatalogId`	The ID of the key catalog to which the key definiton is being added.
`key`	The tag of the key for which the key definition is being added.
`href`	The complete path of the file that the key refers to.
`srcType`	The type of the file that contains the key definition. See the following table for a list of values.
`srcFile`	The complete path of the file that contains the key definition.
`flags`	Bit flags specifying information about the key definition. See the following table for a list of flags.

`srcType` can have one of the following values:

srcType	Meaning
`FV_KeySrcTypeNone`	Source file type not specified.
`FV_KeySrcTypeDitamap`	Source file is a DITA Map.

You can OR the following bit-flags into flags:

Bit mask	Meaning
`FF_DUPLICATE_KEY_DEFINITION`	The specified key definiton is duplicate (that is, it already exists in the key catalog) and will not be used as active definition for resolving keys.
`FF_FOUND_IN_REFERENCED_FILE`	The specified key definiton is contained in a file referenced directly or indirectly from the file that contains the key definition (`srcFile`).
`FF_INVALID_KEY`	The specified key definiton is invalid due to some reason but will still be retained in the key catalog.

Returns

If `F_ApiNewKeyDefinition()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadObjId`	The ID provided does not specify a key catalog.
`FE_BadKey`	The key provided is not valid.
`FE_KeyDefinitionAlreadyExists`	The definition for the specified key is already available in the key catalog and the key definition provided is not duplicate.

F_ApiNewNamedObject()

`F_ApiNewNamedObject()` can create the following named objects:

- `FO_Book`
- `FO_CharFmt`
- `FO_CombinedFontDefn`
- `FO_Color`
- `FO_Command`
- `FO_CondFmt`
- `FO_ElementDef`
- `FO_FmtChangeList`
- `FO_MasterPage`
- `FO_Menu`
- `FO_MenuItemSeparator`
- `FO_PgfFmt`
- `FO_RefPage`
- `FO_RulingFmt`
- `FO_TblFmt`
- `FO_VarFmt`
- `FO_XRefFmt`

`F_ApiNewNamedObject()` uses arbitrary default properties for the objects it creates. After you have created the object, you can use the `F_ApiSet()` functions to change its properties.

IMPORTANT: *When you create a new element definition, it does not appear in the Element Catalog unless you set* `FP_ElementInCatalog` *to* `True`.

IMPORTANT: *When you create a new book and specify a pathname, you must specify an absolute pathname for the* `name` *argument. To create an untitled book, pass an empty string for the name argument.*

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewNamedObject(F_ObjHandleT docId,
    IntT objType,
    StringT name);
```

Arguments

`docId`	The ID of the document to which to add the object
`objType`	The type of object to create (for example, `FO_MasterPage` or `FO_PgfFmt`)
`name`	The name to give to the new object

Returns

The ID of the new named object, or `0` if an error occurs.

If `F_ApiNewNamedObject()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name for the new object is invalid
`FE_BadNew`	Object can't be created
`FE_DupName`	Specified name for the new object belongs to an existing object

Example

The following code creates a new paragraph format named MyPgfFormat and a new master page named MyPg:

```
. . .
F_ObjHandleT docId, pgfFmtId, mstrpgId;

/* Get ID of active document. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Create the paragraph format and master page. */
pgfFmtId = F_ApiNewNamedObject(docId, FO_PgfFmt, "MyPgfFormat");
mstrpgId = F_ApiNewNamedObject(docId, FO_MasterPage, "MyPg");
. . .
```

See also

"F_ApiNewAnchoredObject()" on page 312, "F_ApiNewGraphicObject()" on page 322, and "F_ApiNewSeriesObject()".

F_ApiNewSeriesObject()

`F_ApiNewSeriesObject()` creates a series object. Series objects include the following object types:

- `FO_BodyPage`
- `FO_BookComponent`
- `FO_Pgf`

`F_ApiNewSeriesObject()` allows you to specify the position in the series at which to add the new object.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewSeriesObject(F_ObjHandleT docId,
    IntT objType,
    F_ObjHandleT prevId);
```

Arguments

`docId` The ID of the document to which to add the object.

`objType`	The type of object to create (for example, `FO_BodyPage` or `FO_Pgf`).
`prevId`	The ID of the object in the series after which to add the new object. To add a paragraph at the start of a flow, specify the ID of the flow. To add an object at the beginning of any other series, specify `0`.

Returns

The ID of the new series object, or `0` if an error occurs.

If `F_ApiNewSeriesObject()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadNew`	Object can't be created
`FE_BadObjId`	Invalid object ID
`FE_NotBodyPage`	`prevId` must specify the ID of a body page
`FE_NotPgf`	`prevId` must specify the ID of a paragraph
`FE_NotBookComponent`	`prevId` must specify the ID of a book component

Example

The following code inserts a paragraph after the paragraph containing the insertion point:

```
. . .
F_ObjHandleT docId, pgfId;
F_TextRangeT tr;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (tr.beg.objId == 0) return;

pgfId = F_ApiNewSeriesObject(docId, FO_Pgf, tr.beg.objId);
. . .
```

See also

"F_ApiNewAnchoredObject()" on page 312 and "F_ApiNewNamedObject()" on page 327.

Structured F_ApiNewSubObject()

`F_ApiNewSubObject()` creates the following types of format rule objects:

- `FO_FmtRule`
- `FO_FmtRuleClause`

- FO_FmtChangeList

To create a named format change list (FO_FmtChangeList object) in the format change list catalog, use F_ApiNewNamedObject().

F_ApiNewSubObject() allows you to associate the new object with a specified property of the parent object. For example, you can create an FO_FmtRule as the text format rule of an element definition or as a subformat rule of a format rule clause.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewSubObject(F_ObjHandleT docOrBookId,
    F_ObjHandleT parentId
    IntT property);
```

Arguments

docOrBookId	The ID of the document or book in which to create the new object
parentId	The ID of the object's parent object
property	The property of the parent object to associate the new object with

The following table shows the parent objects for which you can create format rule objects. It also lists the properties with which you can associate the new object, the type of object you can create, and how many objects of that type of object you can associate with each property.

Parent object type	Property	Type of new object	Number that can be created
FO_ElementDef	FP_TextFmtRules	FO_FmtRule	Multiple
	FP_ObjectFmtRules	FO_FmtRule	One
	FP_PrefixRules	FO_FmtRule	Multiple
	FP_SuffixRules	FO_FmtRule	Multiple
	FP_FirstPgfRules	FO_FmtRule	Multiple
	FP_LastPgfRules	FO_FmtRule	Multiple
FO_FmtRule	FP_FmtRuleClauses	FO_FmtRule Clause	Multiple

Parent object type	Property	Type of new object	Number that can be created
FO_FmtRuleClause	FP_SubFmtRule	FO_FmtRule	One
FO_FmtRuleClause	FP_FmtChangeList	FO_FmtChange List	One

If you associate a new format rule object with a property that can specify multiple objects, `F_ApiNewSubObject()` adds the new object after any existing objects for that property.

If you attempt to associate a new format rule object with a property that can specify only one object and that property already has an object associated with it, `F_ApiNewSubObject()` fails, returning `0`.

Returns

The ID of the new format rule object, or `0` if an error occurs.

If `F_ApiNewSubObject()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadNew`	Object can't be created
`FE_BadPropNum`	The property number specified for `property` is invalid
`FE_WrongProduct`	Current FrameMaker product isn't supported

Example

The following code adds a format rule to the Section element definition so that the element definition appears as shown in Figure 2-1:

```
. . .
#define in (MetricT) 65536*72
F_ObjHandleT docId, sectEdefId, fmtRuleId, clauseId;
F_ObjHandleT changeListId;

/* Get ID of Section element definition. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
sectEdefId = F_ApiGetNamedObject(docId, FO_ElementDef,
                "Section");

/* Add the format rule. */
fmtRuleId = F_ApiNewSubObject(docId, sectEdefId,
                FP_TextFmtRules);

/* Set rule type and other characteristics. */
F_ApiSetInt(docId, fmtRuleId, FP_FmtRuleType,
             FV_CONTEXT_RULE);

/* Add rule clause to format rule. */
clauseId = F_ApiNewSubObject(docId, fmtRuleId,
                FP_FmtRuleClauses);
F_ApiSetString(docId, clauseId, FP_ContextLabel,"SubSection");
F_ApiSetString(docId, clauseId, FP_Specification,"Section");

/* Create format change list and add it to rule clause. */
changeListId = F_ApiNewSubObject(docId, clauseId,
                FP_FmtChangeList);
F_ApiSetMetric(docId, changeListId, FP_LeftIndentChange, in);
. . .
```

Element (Container): Section
General rule: Head, (Para | List)+, Section*
Text format rules
1. **If context is:**
Context label: SubSection
Basic properties
Indents
Move left indent by: 1.0"

Figure 2-1 *Section element definition*

See also

"F_ApiNewSeriesObject()" on page 329.

F_ApiNewTable()

`F_ApiNewTable()` inserts a table (`FO_Tbl` object).

When you create a table in the user interface, you can specify a Table Catalog format for the table. The FrameMaker product uses the following properties of the Table Catalog format as the defaults for the new table:

- Number of body rows (`FP_TblInitNumBodyRows`)
- Number of columns (`FP_TblInitNumCols`)
- Number of footer rows (`FP_TblInitNumFRows`)
- Number of header rows (`FP_TblInitNumHRows`)
- Paragraph formats for header, body, and footer cells

For example, if the Table Catalog format's `FP_TblInitNumCols` property is set to `8`, the `FP_NumCols` property of the new table is set to `8`.

With `F_ApiNewTable()`, you can use the Table Catalog format properties as defaults for the number of rows and columns in a new table, or you can provide your own defaults.

After you have created a table, you can add or remove rows with `F_ApiAddRows()` and `F_ApiDeleteRows()`. You can add or remove columns with `F_ApiAddCols()` and `F_ApiDeleteCols()`. You can also use `F_ApiSet()` functions to change the table's other properties.

If you use `F_ApiNewTable()` to create a table in a structured document, FrameMaker applies default element tags, such as Table, Row, and Cell, to the table element and its child elements. To make these elements valid, you must add code to change their tags. In most cases it is easier to add tables to structured documents by calling `F_ApiNewElementInHierarchy()` or `F_ApiNewElement()` to add a table element.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewTable(F_ObjHandleT docId,
    StringT format,
    IntT numCols,
    IntT numBodyRows,
    IntT numHeaderRows,
    IntT numFooterRows,
    F_TextLocT *textLocp);
```

Arguments

`docId`	The ID of the document to which to add the table.
`format`	The table format tag (for example, `FormatA` or `Wide Table`). To use the default format, specify `NULL`.
`numCols`	The number of columns in the table. To use the default number of columns from the Table Catalog format, specify `-1`.
`numBodyRows`	The number of rows in the table. To use the default number of body rows from the Table Catalog format, specify `-1`.
`numHeaderRows`	The number of heading rows in the table. To use the default number of header rows from the Table Catalog format, specify `-1`.
`numFooterRows`	The number of footing rows in the table. To use the default number of footer rows from the Table Catalog format, specify `-1`.
`textLocp`	The location at which to insert the new table. The location can't be within a footnote or a table.

Returns

The ID of the new table, or `0` if an error occurs.

If `F_ApiNewTable()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID.
`FE_BadObjId`	Invalid object ID.
`FE_NotTextObject`	Object specified for the text location isn't a paragraph (`FO_Pgf`).
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified paragraph or text line.

FA_errno value	Meaning
FE_BadOperation	Function call specified an illegal operation.
FE_BadNew	Table can't be created; the format specified by format may not exist or the text location specified by textLocp is in a table or a footnote.

Example

The following code inserts a new table at the insertion point. The new table uses all the defaults from the Format A Table Catalog format, except the number of heading rows, which is 0.

```
. . .
F_ObjHandleT docId, tblId;
F_TextRangeT tr;

/* Get the insertion point. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Insert the table at the insertion point. */
tblId = F_ApiNewTable(docId, "Format A", -1, -1,0, -1, &tr.beg);
. . .
```

See also

"F_ApiAddCols()" on page 70, "F_ApiAddRows()" on page 78, and "F_ApiNewAnchoredObject()" on page 312.

F_ApiNewXML()

F_ApiNewXML() creates a new, untitled XML.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiNewXML(const F_PropValsT *newXMLParams,
F_PropValsT **newXMLReturnParamspp);
. . .
```

Arguments

`newXMLParams`	A property list that instructs FrameMaker how to open the file and how to respond to errors and other conditions. To use the default list, specify `NULL`.
`newXMLReturnParamspp`	A property list that returns the filename and provides information about how the FrameMaker product opened the file. It must be initialized before you call `F_ApiNewXML()`.

To get a property list to specify for the `newXMLParams` parameter, use `F_ApiGetNewXMLDefaultParams()`.

Returns

ID of the new XML document.

Example

The following code demonstrates how to create a new XML file with a specified structured application.

```
F_PropValsT params = F_ApiGetNewXMLDefaultParams ();
F_PropValsT *retParams = NULL;
for (UIntT i = 0; i < params.len; i++)
{
    switch (params.val[i].propIdent.num)
    {
        case FS_StructuredApplication:
            params.val[i].propVal.u.sval = F_StrCopyString
("My_Strcutured_App");
            break;
    }
}
F_ApiNewXML (&params, &retParams);
F_ApiDeallocatePropVals (&params);
F_ApiDeallocatePropVals (retParams);
```

F_ApiNotification()

`F_ApiNotification()` requests that the FrameMaker product notify your client whenever a specified event, or stage of an event, occurs.

IMPORTANT: *If the FrameMaker product encounters an internal error and exits, it does not send any notification to your client about operations performed after the error occured. For example, after an error the product allows the user to save changes in open documents, but it does not notify any clients of the save operations.*

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiNotification(IntT notification,
       IntT state);
```

Arguments

`notification`	Constant that specifies the notification point. See the following table for a list of available constants.
`state`	Specifies whether to turn notification on or off. `True` turns it on, and `False` turns it off.

Many events have several notification points or stages that you can request notification for. The following table lists the notification points and the constants that specify them:

Event or operation	Notification points	Notification constants
Frame binary document opened	Before checking the type of the file to be opened	`FA_Note_PreFileType`
	After checking the type of the file to be opened	`FA_Note_PostFileType`
	Before opening the file	`FA_Note_PreOpenDoc`
	After opening the file	`FA_Note_PostOpenDoc`
MIF document opened	Before checking the type of the file to be opened	`FA_Note_PreFileType`
	After checking the type of the file to be opened	`FA_Note_PostFileType`
	Before opening the file	`FA_Note_PreOpenMIF`
	After opening the file	`FA_Note_PostOpenMIF`

Event or operation	Notification points	Notification constants
SGML document opened	Before checking the type of the file to be opened	`FA_Note_PreFileType`
	After checking the type of the file to be opened	`FA_Note_PostFileType`
	Before opening the file	`FA_Note_PreOpenSGML`
	After opening the file	`FA_Note_PostOpenSGML`
XML document opened	Before checking the type of the file to be opened	`FA_Note_PreFileType`
	After checking the type of the file to be opened	`FA_Note_PostFileType`
	Before opening the file	`FA_Note_PreOpenXML`
	After opening the file	`FA_Note_PostOpenXML`
Filterable document opened	Before checking the type of the file to be opened	`FA_Note_FilterIn`
Frame binary book opened	Before checking the type of the file to be opened	`FA_Note_PreFileType`
	After checking the type of the file to be opened	`FA_Note_PostFileType`
	Before opening the file	`FA_Note_PreOpenBook`
	After opening the file	`FA_Note_PostOpenBook`
MIF book opened	Before checking the type of the file to be opened	`FA_Note_PreFileType`
	After checking the type of the file to be opened	`FA_Note_PostFileType`
	Before opening the file	`FA_Note_PreOpenBookMIF`
	After opening the file	`FA_Note_PostOpenBookMIF`
User double-clicked to open a document in a book window	Before opening the file	`FA_Note_PreBook ComponentOpen`
	After opening the file	`FA_Note_PostBook ComponentOpen`

Event or operation	Notification points	Notification constants
Generating a list or TOC for a document or a book	Before generating the file	`FA_Note_PreGenerate`
	After generating the file	`FA_Note_PostGenerate`
Document saved in Frame binary format	Before saving the document	`FA_Note_PreSaveDoc`
	After saving the document	`FA_Note_PostSaveDoc`
Document saved in MIF format	Before saving the MIF file	`FA_Note_PreSaveMIF`
	After saving the MIF file	`FA_Note_PostSaveMIF`
Document saved as PDF	Before specifying Acrobat settings and generating PostScript	`FA_Note_PreSaveAs PDFDialog`
	After specifying Acrobat settings and generating PostScript	`FA_Note_PostSaveAs PDFDialog`
	Before distilling the PostScript	`FA_Note_PreDistill`
	After distilling the PostScript	`FA_Note_PostDistill`
Document saved as filterable type	Before the document is saved	`FA_Note_FilterOut`
Document exited	Before exiting the document	`FA_Note_PreQuitDoc`
	After exiting the document	`FA_Note_PostQuitDoc`
Book exited	Before exiting the book	`FA_Note_PreQuitBook`
	After exiting the book	`FA_Note_PostQuitBook`
First change made to a document since it was opened or saved	After the document is changed	`FA_Note_DirtyDoc`
First change made to a book since it was opened or saved	After the book is changed	`FA_Note_DirtyBook`

Event or operation	Notification points	Notification constants
Book saved in Frame binary format	Before saving the book	`FA_Note_PreSaveBook`
	After saving the book	`FA_Note_PostSaveBook`
Book saved in MIF format	Before saving the MIF file	`FA_Note_PreSaveBookMIF`
	After saving the MIF file	`FA_Note_PostSaveBookMIF`
Document saved with Autosave	Before saving the document	`FA_Note_PreAutoSaveDoc`
	After saving the document	`FA_Note_PostAutoSaveDoc`
Document reverted	Before reverting the document	`FA_Note_PreRevertDoc`
	After reverting the document	`FA_Note_PostRevertDoc`
Book reverted	Before reverting the book	`FA_Note_PreRevertBook`
	After reverting the book	`FA_Note_PostRevertBook`
FrameMaker product exited	Before the OK to Exit dialog box appears	`FA_Note_PreQuitSession`
	Immediately before exiting the session	`FA_Note_PostQuitSession`
Another client calls `F_ApiCallClient( )` with `clname` set to the current client's name	After `F_ApiCallClient( )` is called	`FA_Note_ClientCall`
Any user action, such as a command choice or text entry	After the FrameMaker product finishes processing the user action	`FA_Note_BackToUser`
Text inset owned by current client clicked	After the user clicked the inset	`FA_Note_DisplayClientTi Dialog`
FrameMaker product updating all text insets	When the client needs to update insets that belong to it	`FA_Note_UpdateAllClientTi`

Event or operation	Notification points	Notification constants
FrameMaker product updating a specific text inset	When the client needs to update a specified inset	`FA_Note_UpdateClientTi`
Text or graphic imported	Before importing the text or graphic	`FA_Note_PreImport`
	After importing the text or graphic	`FA_Note_PostImport`
FrameMaker product command invoked or text entered in a document	Before the FrameMaker product executes command or adds text to the document	`FA_Note_PreFunction`
	After the FrameMaker product executes command or adds text to the document	`FA_Note_PostFunction`
Mouse button clicked	Before the FrameMaker product responds to the mouse click	`FA_Note_PreMouseCommand`
	After the FrameMaker product responds to the mouse click	`FA_Note_PostMouseCommand`
Hypertext command invoked	Before the FrameMaker product executes the hypertext command	`FA_Note_PreHypertext`
	After the FrameMaker product executes the hypertext command	`FA_Note_PostHypertext`
The user clicked Go To Source in the cross-reference dialog box	Before the FrameMaker product goes to the cross-reference source	`FA_Note_PreGoToXrefSrc`
	After the FrameMaker product goes to the cross-reference source	`FA_Note_PostGoToXrefSrc`

Event or operation	Notification points	Notification constants
Document or book printed	After the user clicks OK in the Print dialog box, but before the FrameMaker product prints the document or book	`FA_Note_PrePrint`
	After the FrameMaker product prints the document or book	`FA_Note_PostPrint`
Body page added to document	After the FrameMaker product adds the body page	`FA_Note_BodyPageAdded`
Body page deleted from document	After the FrameMaker product deletes the body page	`FA_Note_BodyPageDeleted`
Structural element inserted[a]	Before the element is inserted	`FA_Note_PreInsertElement`
	After the element is inserted	`FA_Note_PostInsertElement`
Structural element copied	Before the element is copied.	`FA_Note_PreCopyElement`
	After the element is copied.	`FA_Note_PostCopyElement`
Structural element changed	Before the element is changed	`FA_Note_PreChangeElement`
	After the element is changed	`FA_Note_PostChangeElement`
Structural element wrapped	Before the element is wrapped	`FA_Note_PreWrapElement`
	After the element is wrapped	`FA_Note_PostWrapElement`
Structural element dragged	Before the element is dragged	`FA_Note_PreDragElement`
	After the element is dragged	`FA_Note_PostDragElement`
An attribute value is set	Before the attribute value is set	`FA_Note_PreSetAttrValue`
	After the attribute value is set	`FA_Note_PostSetAttrValue`

Event or operation	Notification points	Notification constants
Element definitions are imported	Before the element definitions are imported	`FA_Note_PreImportElementDefs`
	After the element definitions are imported	`FA_Note_PostImportElementDefs`
Inline input of double-byte text	Before the text entry	`FA_Note_PreInlineTypeIn`
	After the text entry	`FA_Note_PostInlineTypeIn`
An inset is selected	When the inset is selected	`FA_Note_U3DCommand`: if the selected inset is a u3d object `FA_Note_Not_U3DCommand`: if the selected inset is not a u3d object
A Robohelp screen capture is inserted	When the screen capture is inserted	`FA_Note_RSC_Supported_File`: When the file supports Robohelp screen capture. `FA_Note_Not_RSC_Supported_File` : When the file does not support Robohelp screen capture.
A graphic supported by Adobe Illustrator (AI) is clicked. Following format are supported by AI: "FrameImage", "FrameVector", "PDF", "EPSI", "TIFF", "SVG", "DIB", "EMF", "JPEG", "QuickDraw PICT", "PNG", "PSD", "WMF", "DXF	When the graphic is clicked	`FA_Note_AI_Supported_File`
A graphic not supported by Adobe Illustrator is clicked.	When the graphic is clicked	`FA_Note_Not_AI_Supported_File`
The workspace launches a particular client's modeless dialog	Before the modeless dialog is launched	`FA_Note_Dialog_Create`

Event or operation	Notification points	Notification constants
When a client's modeless dialog is closed because of workspace-related events such as switching workspaces	Before the modeless dialog is closed	`FA_Note_QuitModelessDialog`
When a client displays or updates (in case the dialog is already displayed) a cross-reference dialog	Before the dialog is displayed or updated.	`FA_Note_DisplayClientXRefDialog`
A particular DITA reference is updated as specified using the element ID and the reference type string	After the reference is updated	`FA_Note_UpdateDITAReference`
All DITA references of the specified type are updated.	After the references are updated	`FA_Note_UpdateDITAReferences`
A DITA map is published	Before publishing starts	`FA_Note_PrePublishDitamap`
A DITA map is published	After the book or document has been generated	`FA_Note_PostPublishDitamap`
When FrameMaker wants to determine whether a command is enabled or disabled		`FA_Note_IsCommandEnabled`
When a view is switched	Before theview is switched	`FA_Note_PreSwitchView`
When a view is switched	After the view is switched	`FA_Note_PostSwitchView`
Key catalog information is not loaded and hence cannot be used.		`FA_Note_LoadKeyCatalog`
Key catalog information is not up-to-date. Hence, the Key catalog needs to be re-loaded before being used.		`FA_Note_ReLoadKeyCatalog`

Event or operation	Notification points	Notification constants
Export		FA_Note_PreExport
		FA_Note_PostExport
		FA_Note_FilterFileToFile
		FA_Note_Dialog
		FA_Note_Alert
		FA_Note_Palette
		FA_Note_ToolBar
		FA_Note_ConsoleMessage
		FA_Note_Help
		FA_Note_URL
		FA_Note_CursorChange
		FA_Note_FontSubstitution
		FA_Note_UndoCheckpoint
		FA_Note_FileOpen

a. FrameMaker issues the FA_Note_PreInsertElement and FA_Note_PostInsertElement notifications when the user inserts an element using the element catalog or a menu command or dialog box which inserts a marker, footnote, cross-reference, equation, anchored frame, or table. FrameMaker does not issue the FA_Note_PreInsertElement and FA_Note_PostInsertElement notifications when it automatically inserts elements because the user or a client added rows, columns, or table titles; when the user or a client imports a graphic; or when a client adds an element programmatically.

The notification constants are numbered sequentially, starting with 0. The API provides a constant, FA_Note_Num, that specifies the total number of notifications. This makes it easy to request notification for all notification points, as shown in the following example code.

Returns

FE_Success if it succeeds, or an error code if it fails.

If `F_ApiNotification()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred
`FE_BadNotificationNum`	The specified notification number was invalid

Example

The following code instructs the FrameMaker product to notify your client for all notification points:

```
. . .
IntT i;
for (i=0; i < FA_Note_Num; i++)
    F_ApiNotification(i, True);
. . .
VoidT F_ApiNotify(notification, docId, sparm, iparm)
    IntT notification;
    F_ObjHandleT docId;
    StringT sparm;
    IntT iparm;
{
    F_Printf(NULL, "Notification: %d:\n", notification);
}
. . .
```

See also

"F_ApiNotify()" on page 349.

F_ApiNotify()

`F_ApiNotify()` is a callback that you can define in your client. The FrameMaker product calls `F_ApiNotify()` when the user or another API client initiates an operation, such as Open or Save, for which your client has requested notification. To request notification, you must call `F_ApiNotification()` for each notification point.

Your client can cancel any command or action for which it receives a `FA_Note_Pre`*`NotificationPoint`* notification. For example, if it receives the `FA_Note_PreQuitDoc` notification, it can cancel the Close command and prevent the user from closing a document. To abort a command, call `F_ApiReturnValue()`,

with the parameter set to `FR_CancelOperation`, when your client receives notification for the command. For more information, see "Canceling commands" on page 221 of the *FDK Programmer's Guide*.

IMPORTANT: *If the FrameMaker product encounters an internal error and exits, it does not send any notification to your client about operations performed after the error occured. For example, after an error the product allows the user to save changes in open documents, but it does not notify any clients of the save operations.*

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiNotify(IntT notification,
    F_ObjHandleT docId,
    StringT sparm,
    IntT iparm);
```

Arguments

`notification`	A constant that indicates the event and the notification point. See the table in "F_ApiNotification()" on page 339 for a list of the constants.
`docId`	The ID of the active document when the event occurs. For filters, the document into which the filter should import its data; if this is zero, the filter must create a new document.
`sparm`	The string, if any, associated with the notification. For example, if the notification is for an Open or Save operation, `sparm` specifies the pathname of the affected file. If the notification is for text entry, `sparm` specifies the text the user typed.
`iparm`	If `notification` is `FA_NotePreFunction` or `FA_NotePostFunction`, `iparm` is the f-code for the command. If `notification` is `FA_NotePreMouseCommand` or `FA_NotePostMouseCommand`, `iparm` specifies bit flags specifying the number of mouse clicks and the modifier keys the user was holding down. See the table below for a list of the flags.

IMPORTANT: For book-wide commands, the FrameMaker product posts an FA_NotePreFunction and FA_NotePostFunction notification for the book file, and for each document in the book. When trapping book-wide functions, you should check the value of `docId` to determine whether it indicates a document or the active book.

For example, if you search a book with two documents in it, the FrameMaker product posts the following function notifications:

FA_Note_PreFunction (start searching book)
FA_Note_PreFunction (start searching first document)
FA_Note_PostFunction (stop searching first document)
FA_Note_PreFunction (start searching second document)
FA_Note_PostFunction (stop searching second document))
FA_Note_PostFunction (stop searching book)

The following table shows the values sparm has for specific notifications.

Notifications	sparm value
All Open, Save, Print, and Close notifications for documents and books	The complete pathname of the document or book.
FA_Note_PreDistill FA_Note_PostDistill	When this notification occurs as a result of saving a document or book as PDF, sparm contains the complete pathname of thePostScript file that was generated from the document or book
FA_Note_PreFunction FA_Note_PostFunction	If the user typed text, the text. If the user applied a paragraph or character format or a font family, the name of the format or font family.
FA_Note_PreChangeElement FA_Note_PostChangeElement	The element tag of the changed structural element.
FA_Note_PreGenerate FA_Note_PostGenerate	The book's filename. If the book is untitled, NULL.
FA_Note_PreHypertext	The string of the hypertext command.
FA_Note_PostHypertext	Null
FA_Note_PreImport FA_Note_PostImport	The name of the imported file.
FA_Note_PreInsertElement	The element tag of the inserted structural element.
FA_Note_PostInsertElement	The element tag of the inserted structural element.
FA_Note_PreOpenBook FA_Note_PostOpenBook	The complete pathname of the book file. If the book is untitled, NULL.
FA_Note_PreSetAttrValue FA_Note_PostSetAttrValue	The attribute name.
FA_Note_PreWrapElement	The element tag of the wrapped structural element.
FA_Note_PostWrapElement	The element tag of the wrapped structural element.

Notifications	sparm value
FA_Note_UpdateDITAReference	DITA Reference type string: • DITA_AUTO • DITA_CONREF • DITA_XREF • DITA_LINK • DITA_TOPICREF • DITA_TOPICSETREF
FA_Note_UpdateDITAReferences	NULL
FA_Note_PreSwitchView	The name of the new view.
FA_Note_PostSwitchView	The name of the old view.
All other notifications	NULL.

The following table shows the values `iparm` has for specific notifications.

Notifications	iparm value
FA_Note_PreFunction FA_Note_PostFunction	The f-code for the command the user invoked. For a list of f-code constants, see the `fcodes.h` header file provided with the FDK.
FA_Note_UpdateAllClientTi	One of the following flags to indicate which text insets are to be updated: `FV_UpdateAllAutomaticClientTi` indicates all insets with the `FP_TiAutomaticUpdate` property set to `True`. `FV_UpdateAllManualClientTi` indicates all insets with the `FP_TiAutomaticUpdate` property set to `False`. `FV_UpdateAllClientTi` indicates all insets regardless of the setting for the `FP_TiAutomaticUpdate` property.
FA_Note_UpdateClientTi	The ID of the text inset.
FA_Note_PreHypertext FA_Note_PostHypertext	The ID of the hypertext object that was activated. If the hypertext message is for an FDK client, `iparm` is the same as the ID passed to the `objId` parameter of the client's `F_ApiMessage()` callback.
FA_Note_PreGoToXrefSrc FA_Note_PostGoToXrefSrc	The ID of the associated cross-reference

Notifications	iparm value
FA_Note_PreMouseCommand FA_Note_PostMouseCommand	The 8 high bits specify the number of mouse clicks minus one. The next 8 bits indicate the modifier key used: FF_SHIFT_KEY, FF_CONTROL_KEY, FF_ALT_KEY, or FF_CMD_KEY The 16 low bits specify the mouse action: for example, FF_TEXT_SEL if the user is selecting text; FF_TEXT_EXT if the user is extending an existing selection; or OBJ_SEL if the user is selecting an object. For a complete list of the constants for mouse actions, see /* Mouse actions */ in the fapidefs.h header file provided with the FDK.
FA_Note_PreInsertElement FA_Note_PostInsertElement	The ID of the inserted structural element.
FA_Note_PreWrapElement FA_Note_PostWrapElement	The ID of the newly created structural element.
FA_Note_PreDragElement	The ID of the structural element that is cut if the operation is completed.
FA_Note_PostDragElement	The ID of the structural element that is pasted if the operation is completed.
FA_Note_PreSetAttrValue FA_Note_PostSetAttrValue	The ID of the element for which the attribute is set.
FA_Note_PreImportElemDefs	The ID of the document from which element definitions are imported.
FA_Note_PreSaveAsPDFDialog FA_Note_PostSaveAsPDFDialog	When this notification occurs as a result of the user choosing to save as PDF in the Save As dialog box, the value is non-zero. When this notification is the result of the API saving as PDF, the value is zero.
FA_Note_UpdateDITAReference	The ID of the element to be updated

Notifications	iparm value
`FA_Note_UpdateDITAReferences`	The `refType` for which DITA references will be updated. The refType value can be one of: • `FV_DITAObjTypeConref`: Update all references of type DITA Conref. • `FV_DITAObjTypeXref`: Update all references of type DITA Xref. • `FV_DITAObjTypeLink`: Update all references of type DITA Link. • `FV_DITAObjTypeTopicref`: Update all references of type DITA Topicref. • `FV_DITAObjTypeTopicsetref`: Update all references of type DITA Topicsetref.
All other notifications	`0`.

Returns

`VoidT`.

Example

See “Adding the F_ApiNotify() callback” on page 220 of the *FDK Programmer’s Guide*.

See also

“F_ApiNotification()” on page 339.

F_ApiObjectValid()

`F_ApiObjectValid()` determines if an object ID identifies an object in the specified document. An object ID is invalid if the object has been deleted. In general, an object ID is valid for only one document in a session, so `F_ApiObjectValid()` is useful for debugging your clients.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiObjectValid(F_ObjHandleT docId,
   F_ObjHandleT objId);
```

Arguments

`docId`	The ID of the document containing the object
`objId`	The ID whose validity you want to determine

Returns

`True` if the ID identifies an object in the current document, or `False` if it doesn't.

If `F_ApiObjectValid()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID

Example

The following code determines if an ID is a valid ID for an object in the current document, and then removes the object:

```
. . .
F_ObjHandleT docId, objId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
objId = F_ApiGetNamedObject(docId, FO_PgfFmt, "Body");

/* Code that might remove object or invalidate ID here. */

if (F_ApiObjectValid(docId, objId))
    F_ApiDelete(docId, objId);
. . .
```

F_ApiOpen()

`F_ApiOpen()` opens a document or book. It can also create a new document.

`F_ApiOpen()` allows you to specify a script (property list) telling the FrameMaker product how to open or create the file and how to deal with error and warning conditions. For example, you can specify whether to abort or to continue opening a document if it contains fonts that are not available. If the file is already open and invisible, it will make the file visible. If you are using FileNames that have characters above ASCII 128 see the FrameMaker FAQ on the Developer Web Page.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiOpen(StringT fileName,
    F_PropValsT *openParamsp,
    F_PropValsT **openReturnParamspp);
```

Arguments

`fileName`	The full pathname of the file to open. If you are using `F_ApiOpen()` to create a document, specify the name of the template to use. For information on how filenames and paths on different platforms are expressed, see the *FDK Platform Guide* for that platform.
`openParamsp`	A property list telling the FrameMaker product how to open the file and how to respond to errors and other conditions. To use the default list, specify `NULL`.
`openReturnParamspp`	A property list that returns the filename and provides information about how the FrameMaker product opened the file. It must be initialized before you call `F_ApiOpen()`.

IMPORTANT: *Always initialize the pointer to the property list that you specify for* `openReturnParamspp` *to* `NULL` *before you call* `F_ApiOpen()`.

To get a property list to specify for the `openParamsp` parameter, use `F_ApiGetOpenDefaultParams()` or create the list from scratch. For a list of all the properties an Open script property list can include, see "F_ApiGetOpenDefaultParams()" on page 219. For an example of how to create a property list from scratch, see "Example" on page 243 of the *FDK Programmer's Guide*.

To create a new document with `F_ApiOpen()`, set the `FS_NewDoc` property in the `openParamsp` property list to `True`.

IMPORTANT: *When creating a new document (*`FS_NewDoc` *is* `True`*), and you display the New dialog box (*`FS_ShowBrowser` *is* `True`*), the user may click Portrait, Custom, or Landscape. If this occurs,* `F_ApiOpen()` *returns* `0` *and sets* `FA_errno` *to* `FE_WantsPortrait`*,* `FE_WantsCustom`*, or* `FE_WantsLandscape`*. To create a portrait, custom, or landscape document, use* `F_ApiCustomDoc()` *(see "F_ApiCustomDoc()" on page 119.)*

Returns

The ID of the document or book if it opens it successfully, or `0` if an error occurs.

The property list that `openReturnParamspp` is set to has the properties shown in the following table.

Property	Meaning and possible values
`FS_OpenedFileName`	A string that specifies the opened file's pathname. If you scripted `FS_ShowBrowser`, or the file was filtered, or you didn't specify the pathname, this pathname can be different from the one you specified in the Open script.
`FS_OpenNativeError`	The error condition; normally the same value as `FA_errno`. If the file is opened successfully, it is set to `FE_Success`. See the table below for a list of possible values.
`FS_OpenStatus`	A bit field indicating what happened when the file was opened. See the table below for a list of possible flags.

Both the `FS_OpenNativeError` property and the `FA_errno` global variable indicate the result of a call to `F_ApiOpen()`. The following table lists the possible

status flags and the FA_errno and FS_OpenNativeError values associated with them.

FS_OpenNativeError and FA_errno values	Possible FS_OpenStatus flags
FE_BadParameter (file wasn't opened)	FV_FileHadStructure: file had structured features, but current FrameMaker product interface is not structured.
	FV_FileAlreadyOpenThisSession: file is already open and script disallowed opening another copy
	FV_BadFileType: file was an executable file or other unreadable type
	FV_BadFileName: specified filename was invalid
	FV_CantNewBooks: script specified a book that didn't exist (the Open operation can't create a new book)
	FV_BadScriptValue: Open script contained an invalid property value
	FV_MissingScript: F_ApiOpen() called without a script
	FV_CantForceOpenAsText: Open script attempted to open the file as text, but file was wrong type
	FV_DisallowedType: file was a Frame binary document and the Open script disallowed it
	FV_DocDamagedByTextFilter: file was a text document and was damaged when it was filtered
	FV_DocHeadersDamaged: the document headers were damaged (probably because of a file system problem)
	FV_DocWrongSize: file is the wrong size (probably because of a file system problem)
	FV_ChecksumDamage: bad checksum
FE_SystemError (file wasn't opened)	FV_TooManyWindows: too many windows were open
	FV_BadTemplate: a bad template was specified
	FV_FileNotReadable: don't have read permission for the file

FS_OpenNativeError and FA_errno values	Possible FS_OpenStatus flags
FE_Success (file was opened)	FV_FileHasNewName: filename was changed from the name specified in the F_ApiOpen() call
	FV_RecoverFileUsed: recover file was present, and it was used
	FV_AutoSaveFileUsed: Autosave file was present, and the user or the Open script chose to use it
	FV_FileWasFiltered: file was filterable and it was filtered
	FV_FontsWereMapped: the document contained unavailable fonts, which were mapped to substitute fonts
	FV_FontMetricsChanged: the file contained fonts with changed metrics, but it was opened anyway
	FV_FontsMappedInCatalog: the Paragraph or Character Catalog used unavailable fonts, which were mapped to substitute fonts
	FV_LanguagesWerentFound: the document used some unavailable languages, but it was opened anyway
	FV_FileIsOldVersion: the file was from an old FrameMaker product version, but the user or the Open script chose to open it anyway
	FV_FileStructureStripped: the file had structured features, which the user or the Open script chose to strip
	FV_FileIsText: the file was a Text Only file, but the user or the Open script chose to open it anyway
	FV_OpenedViewOnly: the user or the Open script chose to open the file as a View Only file
	FV_EditableCopyOpened: the file was in use and the user or the Open script opened an editable copy
	FV_BadFileRefsWereMapped: file reference contained illegal characters; the illegal characters were converted to something safe for the current platform
	FV_ReferencedFilesWerentFound: imported graphics files couldn't be found, but the file was opened anyway

FS_OpenNativeError and FA_errno values	Possible FS_OpenStatus flags
FE_Success (file was opened)	
	FV_UnresolvedXRefs: there were unresolved cross-references, but the file was opened anyway
	FV_UnresolvedTextInsets: there were unresolved text insets, but the file was opened anyway
FE_Success, FE_Canceled, FE_FailedState, or FE_CanceledByClient	FV_LockWasReset: file lock was reset
	FV_LockNotReset: file had lock that wasn't reset
	FV_LockCouldntBeReset: file had lock that couldn't be reset
	FV_FileWasInUse: file was in use
	FV_FileIsViewOnly: file is a View Only file
	FV_LockWasInvalid: file had invalid lock
	FV_FileIsNotWritable: The file was not writable, and the user canceled the open via the alert.
	FV_FileModDateChanged: The file has changed since the last time it was opened or saved in the current session.
FE_Canceled (file wasn't opened)	FV_CancelUseRecoverFile: a recover file was present, so the user or the Open script canceled the Open operation
	FV_CancelUseAutoSaveFile: an Autosave file was present, so the user or the Open script canceled the Open operation
	FV_CancelFileIsText: the file was text, so the user or the Open script canceled the Open operation
	FV_CancelFileIsInUse: the file was in use, so the user or the Open script canceled the Open operation
	FV_CancelFileHasStructure: the file had structure, so the user or the script canceled the Open operation
	FV_CancelReferencedFilesNotFound: the file contained referenced files that were not available, so the user or the Open script canceled the Open operation
	FV_CancelLanguagesNotFound: the file contained languages that weren't available, so the user or the Open script canceled the Open operation

FS_OpenNativeError and FA_errno values	Possible FS_OpenStatus flags
	`FV_CancelFontsMapped`: the document contained fonts that needed to be mapped to other fonts, so the user or the Open script canceled the Open operation
	`FV_CancelFontMetricsChanged`: the file contained fonts with changed metrics, so the user or the Open script canceled the Open operation
	`FV_CancelFontsMappedInCatalog`: the document's Character Catalog or Paragraph Catalog contained fonts that needed to be mapped to other fonts, so the user or the Open script canceled the Open operation
	`FV_CancelFileIsDoc`: the file was a document and the Open script disallowed it
	`FV_CancelFileIsMIF`: the file was a MIF file and the Open script disallowed it
	`FV_CancelBook`: the file was a book and the Open script disallowed it
	`FV_CancelBookMIF`: the file was a MIF file and the Open script disallowed it
	`FV_CancelFileIsFilterable`: the file was a filterable file and the Open script disallowed it
	`FV_CancelFileIsOldVersion`: the file was from an old version of a FrameMaker product, so the user or the Open script canceled the Open operation
	`FV_CancelFileIsSgml`: the file was an SGML document and the Open script disallowed it
	`FV_CancelFileIsXML`: the file was an XML document and the Open script disallowed it
	`FV_CancelFileBrowser`: the user canceled the Open operation from the file browser
	`FV_CancelTempDiskFull`: there was insufficient room on the disk to cache data while opening the file.

To determine if a particular `FS_OpenStatus` bit is set, use `F_ApiCheckStatus()`. For more information, see "F_ApiCheckStatus()" on page 96.

After you are done with the property lists you use to call `F_ApiOpen()`, use `F_ApiDeallocatePropVals()` to deallocate the memory they use.

Example

The following code opens a document named `/tmp/my.doc`. It creates a property list that instructs the FrameMaker product to open `my.doc` in Text Only format, regardless of its type. If the FrameMaker product opens the file successfully, the client displays its filename.

```
. . .
#include "futils.h"
F_PropValsT params, *returnp = NULL;
F_ObjHandleT docId;
IntT i;
UCharT msg[256];

/* Allocate memory for the Open script. */
params = F_ApiAllocatePropVals(1);
if(params.len == 0) return;

/* Open file as Text Only regardless of its type. */
params.val[0].propIdent.num = FS_ForceOpenAsText;
params.val[0].propVal.valType = FT_Integer;
params.val[0].propVal.u.ival = True;

/* Open /tmp/my.doc. */
docId = F_ApiOpen("/tmp/my.doc", &params, &returnp);

/* Get filename of document from return property list. */
if (docId)
    {
    /* Get index of return property that specifies filename. */
    i = F_ApiGetPropIndex(returnp, FS_OpenedFileName);
    F_Sprintf(msg, "Opened %s",
        returnp->val[i].propVal.u.sval);
    F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
    }

/* Free memory used by Open scripts. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnp);
. . .
```

See also

"F_ApiCheckStatus()" on page 96, "F_ApiGetOpenDefaultParams()" on page 219, "F_ApiPrintOpenStatus()" on page 370, and "F_ApiSimpleOpen()" on page 465.

F_ApiOpenResource()

`F_ApiOpenResource()` opens a client resource. With this function, you also have access to FrameMaker cursors.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiOpenResource(IntT resourceType,
    StringT resourceName);
```

Arguments

`resourceType`	The type of resource to open. To open a dialog resource, specify `FO_DialogResource`.
`resourceName`	The name of the resource to open.

Returns

The ID of the opened resource, or `0` if it can't open the resource.

If `F_ApiOpenResource()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_BadOperation`	Function call specified an illegal operation

Example

For two examples, see "F_ApiModalDialog()" on page 303 and "F_ApiModelessDialog()" on page 305.

See also

"F_ApiModalDialog()" on page 303, "F_ApiModelessDialog()" on page 305, and "F_ApiSetClientDir()" on page 412.

F_ApiPaste()

`F_ApiPaste()` pastes the contents of the FrameMaker product Clipboard into a specified document at the insertion point. Cutting and Pasting objects will cause FrameMaker to create a new UID for the pasted object.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiPaste(F_ObjHandleT docId,
    IntT flags);
```

Arguments

`docId`	The ID of the document to which to paste the selected text.
`flags`	Bit field that specifies how to paste the text and how to handle interactive alerts. For default settings, specify `0`.

If you specify `0` for flags, `F_ApiPaste()` suppresses any interactive alerts or warnings that arise. It also inserts columns to the left of the current columns and rows above the current row.

You can also OR the following values into `flags`.

flags constant	Meaning
`FF_INTERACTIVE`	Prompt user with dialog or alert boxes that arise.
`FF_VISIBLE_ONLY`	Cut only the visible portion of the selection.
`FF_DONT_DELETE_HIDDEN_TEXT`	Don’t replace hidden text.
`FF_DONT_APPLY_ALL_ROWS`	Don’t apply condition setting on the Clipboard to all rows. If whole table is selected and the Clipboard contains condition setting, cancel the paste.
`FF_REPLACE_CELLS`	Replace selected cells with cells on the Clipboard.
`FF_INSERT_BELOW_RIGHT`	Add columns to the right of the current column or below the current row.

When you use `F_ApiPaste()` to paste table cells into a table, it does not work exactly like the interactive Paste command. The interactive Paste command automatically overwrites cells if the Clipboard contains less than an entire row or column. For example, if the insertion point is in a three-column table and the Clipboard contains a single cell, the interactive Paste command overwrites the cell containing the

insertion point with the cell on the Clipboard. If two cells in the table are selected, the Paste command overwrites both of them with the cell on the Clipboard.

By default, `F_ApiPaste()` does not overwrite any cells. If the Clipboard contains less than an entire row or column when you call `F_ApiPaste()`, or of the current selection is less than an entire row, `F_ApiPaste` does nothing and returns `FE_BadSelectionForOperation`. This ensures that your do not inadvertently overwrite any cells. To make `F_ApiPaste()` replace cells with the Clipboard contents, you must call it with the `FF_REPLACE_CELLS` flag set.

The `FF_INTERACTIVE` flag takes precedence over other flags. So, if you specify `FF_INTERACTIVE | FF_DONT_DELETE_HIDDEN_TEXT` and the selection contains hidden text, the FrameMaker product prompts the user and allows the user to choose whether to delete the hidden text. It is illegal to specify `FF_REPLACE_CELLS | FF_INSERT_BELOW_RIGHT`.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiPaste()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadOperation`	Function call specified an illegal operation
`FE_BadDocId`	Invalid document ID
`FE_BadSelectionForOperation`	Current text selection is invalid for this operation
`FE_Canceled`	User canceled the operation

Example

The following code makes sure the insertion point is an insertion point (not a selection), and then pastes the text on the Clipboard to it.

```
. . .
F_TextRangeT tr;
F_ObjHandleT docId;

/* Get text selection. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Make sure selection is an insertion point. */
tr.beg.objId = tr.end.objId;
tr.beg.offset = tr.end.offset;
F_ApiSetTextRange(FV_SessionId, docId, FP_TextSelection, &tr);

/* Paste Clipboard contents. */
F_ApiPaste(docId, 0);
. . .
```

See also

"F_ApiClear()" on page 100, "F_ApiCopy()" on page 115, and "F_ApiCut()" on page 123.

F_ApiPopClipboard()

`F_ApiPopClipboard()` pops the Clipboard stack, moving the entry on the top of the stack to the Clipboard.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiPopClipboard(VoidT);
```

Arguments

None.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiPopClipboard()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred
`FE_BadOperation`	Clipboard stack is empty

Example

The following code executes Copy and Paste operations and then restores the original Clipboard contents:

```
. . .
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
F_ApiPushClipboard();
F_ApiCopy(docId, 0);
F_ApiPaste(docId, 0);
F_ApiPopClipboard();
. . .
```

See also

"F_ApiCopy()" on page 115, "F_ApiCut()" on page 123, "F_ApiPaste()" on page 364, and "F_ApiPushClipboard()" on page 384.

F_ApiPrintFAErrno()

`F_ApiPrintFAErrno()` prints the current API error status, represented by the global variable, `FA_errno`. It is useful for debugging clients.

When an API function fails, it stores an error code in the global variable, `FA_errno`. `FA_errno` retains the error code until another function fails and sets it or until your code explicitly sets it. To determine whether a set of API function calls has failed, initialize `FA_errno` to `FE_Success` once before all the calls and check it once after all the calls.

For example, if you call `F_ApiNotification()` and specify an invalid notification constant, the API sets `FA_errno` to `FE_BadNotificationNum`. If you

subsequently call `F_ApiPrintFAErrno()`, it prints the string `FE_BadNotificationNum`.

For a list of error codes and their meanings, see Chapter 5, “Error Codes.”

`F_ApiPrintFAErrno()` prints to the Frame console..

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPrintFAErrno(VoidT);
```

Arguments

None.

Returns

`VoidT`.

If `F_ApiPrintFAErrno()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

The following code enables automatic change bars for the current document and uses `F_ApiPrintFAErrno()` to check for errors.

```
. . .
F_ObjHandleT docId;
IntT changeBarsOn;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

FA_errno = FE_Success;
changeBarsOn = F_ApiGetInt(FV_SessionId, docId,
                           FP_AutoChangeBars);

/* If previous call succeeds, this call prints FE_Success. */
F_ApiPrintFAErrno();
. . .
```

F_ApiPrintImportStatus()

`F_ApiPrintImportStatus()` prints status flags returned by `F_ApiImport()`. It is useful for debugging your clients.

`F_ApiPrintImportStatus()` prints to Frame console.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPrintImportStatus(F_PropValsT *p);
```

Arguments

`p` The property list that `F_ApiImport()` returns in `importReturnParamspp`

Returns

`VoidT`.

Example

The following code prints the import status after calling `F_ApiImport()`:

```
. . .
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT docId;
F_TextRangeT tr;

params = F_ApiGetImportDefaultParams();
if(params.len == 0) return;
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

F_ApiImport(docId, &tr.beg, "/tmp/frame.doc",
             &params, &returnParamsp);

F_ApiPrintImportStatus(returnParamsp);
. . .
```

See also

"F_ApiImport()" on page 286.

F_ApiPrintOpenStatus()

`F_ApiPrintOpenStatus()` prints status flags returned by `F_ApiOpen()`. It is useful for debugging your clients.

`F_ApiPrintOpenStatus()` prints to the Frame console..

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPrintOpenStatus(F_PropValsT *p);
```

Arguments

`p` The property list that `F_ApiOpen()` returns in `openReturnParamspp`

Returns

`VoidT`.

Example

The following code prints the open status after calling `F_ApiOpen()` to open a document named `/tmp/my.doc`:

```
. . .
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT docId = 0;

params = F_ApiGetOpenDefaultParams();
docId = F_ApiOpen("/tmp/my.doc", &params, &returnParamsp);

if (!docId) return;
F_ApiPrintOpenStatus(returnParamsp);
. . .
```

See also

"F_ApiOpen()" on page 355.

F_ApiPrintPropVal()

`F_ApiPrintPropVal()` prints the value of a specified property. It is useful for debugging your clients.

`F_ApiPrintPropVal()` prints to the Frame console.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPrintPropVal(F_PropValT *p);
```

Arguments

p The property to print

Returns

VoidT.

Example

The following code prints the name of the active document:

```
. . .
F_PropValT prop;
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
prop = F_ApiGetPropVal(FV_SessionId, docId, FP_Name);
F_ApiPrintPropVal(&prop);
. . .
```

See also

"F_ApiGetPropVal()" on page 232.

F_ApiPrintPropVals()

F_ApiPrintPropVals() prints the values in a specified property list. It is useful for debugging your clients.

F_ApiPrintPropVals() prints to the Frame console.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPrintPropVals(F_PropValsT *p);
```

Arguments

p The property list

Returns

VoidT.

Example

The following code gets the properties for the paragraph containing the insertion point and prints them to the console:

```
. . .
F_PropValsT props;
F_TextRangeT tr;
F_ObjHandleT docId, pgfId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

props = F_ApiGetProps(docId, tr.end.objId);
F_ApiPrintPropVals(&props);
. . .
```

F_ApiPrintSaveStatus()

`F_ApiPrintSaveStatus()` prints errors returned by `F_ApiSave()`. It is useful for debugging your clients.

`F_ApiPrintSaveStatus()` prints to the Frame console.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPrintSaveStatus(F_PropValsT *p);
```

Arguments

p The property list that `F_ApiSave()` returns in `saveReturnParamspp`

Returns

VoidT.

Example

The following code prints the save status after calling `F_ApiSave()` to save the active document as `/tmp/my.doc`:

```
. . .
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT mydocId;

/* Allocate Save scripts. */
params = F_ApiGetSaveDefaultParams();
if(params.len == 0) return;

/* Save document. */
mydocId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
F_ApiSave(mydocId, "/tmp/my.doc", &params, &returnParamsp);

/* Print save status. */
F_ApiPrintSaveStatus(returnParamsp);
. . .
```

See also

"F_ApiSave()" on page 399.

F_ApiPrintTextItem()

`F_ApiPrintTextItem()` prints the text in a specified text item. It is useful for debugging clients.

`F_ApiPrintTextItem()` prints to the Frame console.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPrintTextItem(F_TextItemT *textItem);
```

Arguments

`textItem`	The text item to print

Returns

`VoidT`.

Example

The following code prints the first string text item in the paragraph containing the insertion point:

```
. . .
F_TextItemsT tis;
F_ObjHandleT docId;
F_TextRangeT tr;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

tis = F_ApiGetText(docId, tr.beg.objId, FTI_String);
if(tis.len == 0) return;

F_ApiPrintTextItem(&tis.val[0]);
F_ApiDeallocateTextItems(&tis);
. . .
```

See also

"F_ApiGetText()" on page 249 and "F_ApiPrintTextItems()" on page 374.

F_ApiPrintTextItems()

`F_ApiPrintTextItems()` prints the text in a specified set of text items (`F_TextItemsT` structure). It is useful for debugging clients.

`F_ApiPrintTextItems()` prints to the Frame console.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPrintTextItems(F_TextItemsT *textItems);
```

Arguments

`textItems`	The set of text items to print

Returns

`VoidT`.

Example

The following code prints the string text items in the main flow in the active document:

```
. . .
F_TextItemsT tis;
F_ObjHandleT docId, flowId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
flowId = F_ApiGetId(FV_SessionId, docId, FP_MainFlowInDoc);

tis = F_ApiGetText(docId, flowId, FTI_String);
if(tis.len == 0) return;

F_ApiPrintTextItems(&tis);
F_ApiDeallocateTextItems(&tis);
. . .
```

See also

"F_ApiGetText()" on page 249 and "F_ApiPrintTextItem()" on page 373.

F_ApiPrintUpdateBookStatus()

`F_ApiPrintUpdateBookStatus()` prints errors returned by `F_ApiUpdateBook()`. It is useful for debugging your clients.

`F_ApiPrintUpdateBookStatus()` prints to the Frame console.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPrintUpdateBookStatus(F_PropValsT *p);
```

Arguments

`p` The property list that `F_ApiUpdateBook()` returns in `updateReturnParamspp`

Returns

`VoidT`.

Example

The following code prints the update status after calling `F_ApiUpdateBook()` to update the active book:

```
. . .
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT bookId;

/* Allocate update scripts. */
params = F_ApiGetUpdateBookDefaultParams();
if(params.len == 0) return;

/* Update book. */
bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook;
F_ApiUpdateBook(bookId, &params, &returnParamsp);

/* Print update status. */
F_ApiPrintUpdateBookStatus(returnParamsp);
. . .
```

See also

"F_ApiSave()" on page 399.

F_ApiProgressBarEx()

`F_ApiProgressBarEx()` runs a progress bar in a separate thread while the user can continue working in the background.

Synopsis

```
#include "fapi.h"
...
StatusT
F_ApiProgressBarEx(BoolT bShow, const F_PropValsT *vals);
```

Arguments

`bShow`	One of the following: `True` (Start the progress bar) `False` (End the progress bar)
`vals`	Structure that includes an array of property-value pairs.

Property	Type	Meaning
`FP_DbTitleLabel`	`StringT`	Sets the title of the progress bar dialog.

Returns

`FE_Success`.

Structured F_ApiPromoteElement()

`F_ApiPromoteElement()` promotes the selected structural element. The selected element becomes a sibling of its former parent. It appears immediately after its former parent. The siblings that follow it become its children.

IMPORTANT: *One structural element must be selected when* `F_ApiPromoteElement()` *is called. The element cannot be the root element or a child of the root element.*

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPromoteElement(F_ObjHandleT docId);
```

Arguments

`docId`	The ID of the document containing the selected structure element

Returns

`VoidT`.

If `F_ApiPromoteElement()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product isn't supported
`FE_BadDocId`	Invalid document ID
`FE_BadSelectionForOperation`	Current selection is invalid for this operation

Example

The following code promotes the selected structural element in the active document:

```
. . .
F_ApiPromoteElement(F_ApiGetId(0, FV_SessionId, FP_ActiveDoc));
. . .
```

See also

"F_ApiDemoteElement()" on page 154.

F_ApiPromptInt()

`F_ApiPromptInt()` displays a dialog box that prompts the user for a single integer value. It allows you to provide a default value, which appears in the entry field when the dialog box appears. The dialog box has OK and Cancel buttons. `F_ApiPromptInt()` does not assign a value to `*intp` if the user clicks Cancel. If the user types alphabetic text after a number, the API ignores the text and just returns the value of the number.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiPromptInt(IntT* intp,
       StringT message,
       StringT stuffVal);
```

Arguments

`intp`	The value in the input field when the user clicks OK.
`message`	The message that appears in the dialog box. It must be 255 characters or less.
`stuffVal`	The default value that appears in the input field when the dialog box is first displayed.

Returns

`0` if the user clicked OK, or a nonzero value if the user clicked Cancel or an error occurred.

If `F_ApiPromptInt()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

The following code displays the dialog box shown in Figure 2-2:

```
. . .
#include "futils.h"

IntT err;
IntT ires;
UCharT mesg[256];
err = F_ApiPromptInt(&ires, "Integer?", "1234");
if (err)
       F_Sprintf(mesg, "Cancel was pressed, ires is undefined");
   else F_Sprintf(mesg, "The value of ires is %d.", ires);
F_ApiAlert(mesg, FF_ALERT_CONTINUE_NOTE);
. . .
```

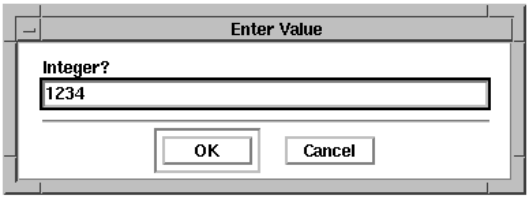


Figure 2-2 *Integer prompt dialog box*

See also

“F_ApiPromptMetric()”, and “F_ApiPromptString()” on page 382.

F_ApiPromptMetric()

`F_ApiPromptMetric()` displays a dialog box that prompts the user for a single metric value. It allows you to provide a default value, which appears in the entry field when the dialog box appears. The dialog box has OK and Cancel buttons. `F_ApiPromptMetric()` does not assign a value to `*metricp` if the user clicks Cancel.

`F_ApiPromptMetric()` dialog boxes behave like metric dialog boxes in the user interface. If the user types a number followed by a string that represents a unit (for example `10pts` or `5"`), the API converts the number into the equivalent number of metric units. If the user doesn't specify a unit, the API uses points (metric 65536).

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiPromptMetric(MetricT *metricp,
    StringT message,
    StringT stuffVal,
    MetricT defaultunit);
```

Arguments

`metricp`	The value in the input field when the user clicks OK.
`message`	The message that appears in the dialog box. It must be 255 characters or less.
`stuffVal`	The default value that appears in the input field when the dialog box is first displayed.
`defaultunit`	The metric unit to use if the user doesn't specify one.

Returns

`0` if the user clicked OK, or a nonzero value if the user clicked Cancel or an error occurred.

If `F_ApiPromptMetric()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

The following code displays the dialog box shown in Figure 2-3:

```
. . .
#include "futils.h"
#define in (MetricT)(65536*72)
IntT err;
MetricT mres;
UCharT mesg[256];
err = F_ApiPromptMetric(&mres, "Metric?", "12.34 in", in);
if (err)
   F_Sprintf(mesg, "Cancel was pressed, mres is undefined");
else
   F_Sprintf(mesg, "The value of mres is %f inches.", mres/in);
F_ApiAlert(mesg, FF_ALERT_CONTINUE_NOTE);
. . .
```

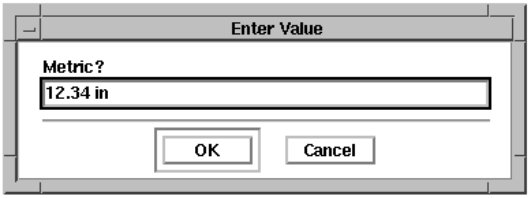


Figure 2-3 *Metric prompt dialog box*

See also

"F_ApiPromptInt()" on page 379 and "F_ApiPromptString()".

F_ApiPromptString()

`F_ApiPromptString()` displays a dialog box that prompts the user for a single string value. It allows you to provide a default string, which appears in the entry field when the dialog box appears. The dialog box has OK and Cancel buttons. `F_ApiPromptString()` does not assign a value to `*stringp` if the user clicks Cancel.

IMPORTANT: *If you are running your client on Windows, do not call* `F_ApiPromptString()` *to prompt the user for a pathname. If the user enters a pathname as a string, the backslash character (\) is interpreted as a special escape character. For example, the characters* `\s` *represent a space. If the user enters the*

pathname `c:\sample`*, this string is interpreted as* `c: ample`*. To prompt the user for a pathname, use* `F_ApiChooseFile()` *to display a file selection dialog box.*

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiPromptString(StringT *stringp,
       StringT message,
       StringT stuffVal);
```

Arguments

`stringp`	The string in the input field when the user clicks OK.
`message`	The message that appears in the dialog box. It must be 255 characters or less. Newline and linefeed characters are ignored.
`stuffVal`	The default value that appears in the input field when the dialog box is first displayed.

IMPORTANT: `F_ApiPromptString()` *allocates memory for the string it returns. Use* `F_Free()` *to free the string when you are done with it.*

Returns

`0` if the user clicked OK, or a nonzero value if the user clicked Cancel or an error occurred.

If `F_ApiPromptString()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

The following code displays the dialog box shown in Figure 2-4:

```
. . .
IntT err;
StringT sres;
err = F_ApiPromptString(&sres, "String?", "Default text");
if (err) return;
F_ApiAlert(sres, FF_ALERT_CONTINUE_NOTE);
F_Free(sres);
. . .
```

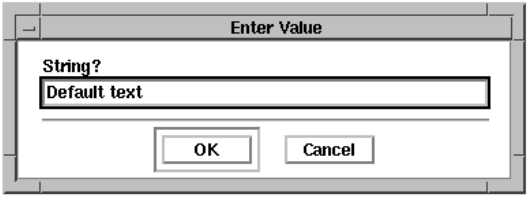


Figure 2-4 *String prompt dialog box*

See also

"F_ApiPromptInt()" on page 379 and "F_ApiPromptMetric()" on page 381.

F_ApiPushClipboard()

`F_ApiPushClipboard()` pushes the current Clipboard contents onto the Clipboard stack. It is useful if you want to use FDK Clipboard functions, such as `F_ApiCopy()` or `F_ApiCut()`, without losing the Clipboard's original contents.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiPushClipboard(VoidT);
```

Arguments

None.

Returns

`VoidT`.

If `F_ApiPushClipboard()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

See "F_ApiPopClipboard()" on page 366.

See also

"F_ApiCopy()" on page 115, "F_ApiCut()" on page 123, "F_ApiPopClipboard()" on page 366, and "F_ApiPaste()" on page 364.

F_ApiQuickSelect()

`F_ApiQuickSelect()` implements a quick-key interface that allows the user to choose a string from a list of strings in the document Tag area.

`F_ApiQuickSelect()` highlights the document Tag area and displays a prompt and the first string in a specified list of strings. The user can display a string in the Tag area by typing the first few letters of the string. The user can also scroll through the strings by pressing the up and down arrow keys. To choose a string, the user presses Return when the string appears in the Tag area. To cancel the choice, the user clicks in the document without pressing Return.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiQuickSelect(F_ObjHandleT docId,
    StringT prompt,
    F_StringsT *stringlist);
```

Arguments

`docId`	The ID of the document containing the Tag area in which to display the prompt
`prompt`	The prompt that appears in the Tag area
`stringlist`	The list of strings from which the user can choose

Returns

An index into the array of strings specified by `stringlist` or `-1` if the user cancels the quick selection.

If `F_ApiQuickSelect()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred
`FE_BadDocId`	The specified document ID is invalid

Example

See “Implementing quick keys” on page 228 of the *FDK Programmer’s Guide*.

F_ApiRedisplay()

`F_ApiRedisplay()` updates the display for a specified document to reflect any changes that occurred while `FP_Displaying` was set to `False`. If you have set `FP_Displaying` to `False` and subsequently reset it to `True`, you should call `F_ApiRedisplay()` to redisplay each document you modified.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiRedisplay(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document to be redisplayed

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiRedisplay()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID

Example

The following code disables redisplaying, and then reenables it and redisplays all the documents in the session:

```
. . .
F_ObjHandleT docId;

F_ApiSetInt(0, FV_SessionId, FP_Displaying, False);

/* Code to change documents without reformatting goes here.  */

F_ApiSetInt(0, FV_SessionId, FP_Displaying, True);

/* Redisplay all the documents in the session. */
docId = F_ApiGetId(0, FV_SessionId, FP_FirstOpenDoc);
while (docId)
    {
    F_ApiRedisplay(docId);
    docId = F_ApiGetId(FV_SessionId, docId,
                        FP_NextOpenDocInSession);
    }
. . .
```

See also

“F_ApiReformat()”.

F_ApiReformat()

`F_ApiReformat()` reformats the specified document. If you have disabled and subsequently reenabled reformatting by setting the session property, `FP_Reformatting`, you should then call `F_ApiReformat()` to reformat each changed document in the session.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiReformat(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document to reformat

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiReformat()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID

Example

The following code disables reformatting, and then reenables it and reformats all the documents in the session:

```
. . .
F_ObjHandleT docId;

F_ApiSetInt(0, FV_SessionId, FP_Reformatting, False);

/* Code to change documents without reformatting goes here.  */

F_ApiSetInt(0, FV_SessionId, FP_Reformatting, True);

/* Reformat all the documents in the session. */
docId = F_ApiGetId(0, FV_SessionId, FP_FirstOpenDoc);
while (docId)
    {
    F_ApiReformat(docId);
    docId = F_ApiGetId(FV_SessionId, docId,
                       FP_NextOpenDocInSession);
    }
. . .
```

F_ApiRehyphenate()

`F_ApiRehyphenate()` rehyphenates a specified document based on changes the user has made to words' hyphenation points.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiRehyphenate(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document to rehyphenate

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiRehyphenate()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Bad document or book ID
`FE_WrongProduct`	Current FrameMaker product doesn't support this operation
`FE_SystemError`	Couldn't allocate memory

Example

The following code rehyphenates the active document:

```
. . .
F_ApiRehyphenate(F_ApiGetId(0, FV_SessionId, FP_ActiveDoc));
. . .
```

F_ApiResetEqnSettings()

`F_ApiResetEqnSettings()` resets the document equation settings to the default settings.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiResetEqnSettings(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document for which to reset equation settings

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiResetEqnSettings()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Bad document or book ID

FA_errno value	Meaning
FE_WrongProduct	Current FrameMaker product doesn't support this operation
FE_SystemError	Couldn't allocate memory

Example

The following code resets the equation settings for the active document:

```
. . .
F_ApiResetEqnSettings(F_ApiGetId(0,FV_SessionId, FP_ActiveDoc));
. . .
```

F_ApiResetReferenceFrames()

F_ApiResetReferenceFrames() resets the reference frames in the specified document. It is useful for updating a document after you have programmatically changed a reference frame that is referenced by paragraphs in the document.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiResetReferenceFrames(F_ObjHandleT docId);
```

Arguments

docId	The ID of the document for which to reset reference frames

Returns

FE_Success if it succeeds, or an error code if an error occurs.

If F_ApiResetReferenceFrames() fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
FE_BadDocId	Bad document or book ID
FE_WrongProduct	Current FrameMaker product doesn't support this operation
FE_SystemError	Couldn't allocate memory

Example

The following code resets reference frames for the active document:

```
. . .
F_ApiResetReferenceFrames(F_ApiGetId(0, FV_SessionId,
                          FP_ActiveDoc));
. . .
```

F_ApiRestartPgfNumbering()

`F_ApiRestartPgfNumbering()` restarts the paragraph numbering for a specified document. For more information on paragraph numbering, see your FrameMaker product user documentation.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiRestartPgfNumbering(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document for which to restart paragraph numbering

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiRestartPgfNumbering()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Bad document or book ID
`FE_WrongProduct`	Current FrameMaker product doesn't support books
`FE_SystemError`	Couldn't allocate memory

Example

The following code restarts paragraph numbering for the active document:

```
. . .
F_ApiRestartPgfNumbering(F_ApiGetId(0, FV_SessionId,
                            FP_ActiveDoc));
. . .
```

F_ApiReturnValue()

`F_ApiReturnValue()` sets a return value for a client-defined callback. It allows a client to provide status information to the FrameMaker product or client that called the callback. You can call it in the following callbacks:

- `F_ApiDialogEvent()`
- `F_ApiNotify()`

`F_ApiReturnValue()` is useful for canceling FrameMaker product operations. When your client receives a `FA_Pre` notification for an operation, it can cancel the operation by calling `F_ApiReturnValue()` with `retval` set to `FR_CancelOperation`. For example, if your client's `F_ApiNotify()` callback responds to all `FA_Note_PrePrint` notifications by calling `F_ApiReturnValue()` with `retval` set to `FR_CancelOperation`, the FrameMaker product cancels all print operations.

Your client can also call `F_ApiReturnValue()` to respond to a `FA_Note_ClientCall` notification. The API returns the value specified by `retval` to the client that called `F_ApiCallClient()`.

Your client can also call `F_ApiReturnValue()` in a `F_ApiDialogEvent()` callback to prevent the FrameMaker product from closing a modal dialog box.

Your client can call `F_ApiReturnValue()` several times in a callback function. The last call it makes before the callback returns overrides any previous calls.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiReturnValue(IntT retval);
```

Arguments

`retval` The value to return

You can set `retval` to any integer. If you client sets `retval` in response to one of the following notifications, it can use the listed constants.

Notification	Values client can pass to F_ApiReturnValue()	Meaning
`FA_Note_DisplayClientTiDialog`	`FR_DisplayedTiDialog`	The client has displayed its version of the Text Inset Properties dialog box
`FA_Note_DisplayClientXRefDialog`	`FR_DisplayedXRefDialog`	If the client displays or updates its cross-reference dialog, it sets the return value as `FR_DisplayedXRefDialog`. If this return value is not set, FrameMaker assumes that the client did not display any dialog and the standard cross-reference dialog is displayed.
`FA_Pre`*`NotificationPoint`*	`FR_CancelOperation`	Cancel the operation for which the notification was issued
`FA_Note_PreSaveAsPDFDialog`	`FR_CancelOperation`	Cancel the Save as PDF operation
	`FR_SkipStep`	Do not display the Acrobat Settings dialog box
`FA_Note_PostSaveAsPDFDialog`	`FR_CancelOperation`	Cancel the Save as PDF operation[a]
`FA_Note_PreDistill`	`FR_CancelOperation`	Cancel the Save as PDF operation
`FA_Note_ClientCall`	Any value recognized by the client that called `F_ApiCallClient()`	Client-defined

Notification	Values client can pass to F_ApiReturnValue()	Meaning
FA_Note_FilterIn	The ID of the document into which the file was filtered	The document was filtered successfully
	0	The document was not filtered successfully
FA_Note_IsCommandEnabled	FR_CommandEnabled	Return value to be used in response to notification FA_Note_IsCommandEnabled if command should be enabled.
FA_Note_IsCommandEnabled	FR_CommandDisabled	Return value to be used in response to notification FA_Note_IsCommandEnabled if command should be disabled

a. Note that this event occurs before the distilling operation begins. You can now cancel the operation after the user closes Save As PDF dialog box.

If your client calls `F_ApiReturnValue()` for notifications other than those listed above, it has no effect.

A client can also call `F_ApiReturnValue()` in a `F_ApiDialogEvent()` callback that responds to actions in a client-defined modal dialog box. Normally, when the user clicks a button in a client-defined modal dialog box, the FrameMaker product calls the client's `F_ApiDialogEvent()` callback and then closes the dialog box. However, if the client's `F_ApiDialogEvent()` callback calls `F_ApiReturnValue()` with `retVal` set to `FR_DialogStayUp`, the FrameMaker product does not close the dialog box. The following table lists the values a client can pass to `F_ApiReturnValue()` in an `F_ApiDialogEvent()` callback.

Values client can pass to F_ApiReturnValue()	Meaning
FR_DialogStayUp	Do not close the modal dialog box in which the event occurred
Any other value	Close the modal dialog box

Returns

The value of the `retval` parameter the previous time `F_ApiReturnValue()` was called in the current callback function.

If `F_ApiReturnValue()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

See "F_ApiCallClient()" on page 93 and "Handling user actions in multiple modeless dialog boxes" on page 454 of the *FDK Programmer's Guide*.

See also

"F_ApiDialogEvent()" on page 155, "F_ApiNotify()" on page 349, and "F_ApiCallClient()" on page 93.

Structured F_ApiRun()

`F_ApiRun()` provides the minimum functionality required in an FDK client's `main()` function.

IMPORTANT: `F_ApiRun()` *is available only to Windows RPC clients.*

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiRun(VoidT);
```

Arguments

None.

Returns

`FE_Success` if it succeeds, or a nonzero value if an error occurs.

C++ clients that need to include their own `main()` functions can call `F_ApiRun()`. `F_ApiRun()` is defined as:

```
IntT F_ApiRun()
{
   StringT s;
   if (s = F_ApiStartUp(NULL))
      F_ApiErr(s);
   else
      while (!FA_bailout)
      F_ApiService(0);
   F_ApiShutDown();
   return(s!=NULL);
}
```

Example

The following code can be used as the main function for an FDK client written in C++:

```
int main() {return (F_ApiRun());}
```

See also

"F_ApiShutDown()" on page 455, "F_ApiService()" on page 405, and "F_ApiErr()" on page 164.

F_ApiSave()

F_ApiSave() saves a document or book. It allows you to script the way the FrameMaker product saves the file and to specify responses to warnings and messages that arise while the file is being saved. You can save a file under its current name or save it as a new file.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiSave(F_ObjHandleT Id,
    StringT saveAsName,
    F_PropValsT *saveParamsp,
    F_PropValsT **saveReturnParamspp);
```

Arguments

Id	The ID of the document or book to save.
saveAsName	The pathname for saving the document or book.
saveParamsp	A property list that tells the FrameMaker product how to save the file and how to respond to errors and other conditions. Use F_ApiGetSaveDefaultParams() or F_ApiAllocatePropVals() to create and allocate memory for this property list. To use the default list, specify NULL.
saveReturnParamspp	A property list that returns information about how the FrameMaker product saved the file.

IMPORTANT: *Always initialize the pointer to the property list that you specify for* saveReturnParamspp *to* NULL *before you call* F_ApiSave().

To get a property list for saveParamsp, you can use F_ApiGetSaveDefaultParams() and modify individual properties in the list it returns, or you can create a list from scratch. For information on F_ApiGetSaveDefaultParams(), see "F_ApiGetSaveDefaultParams()" on page 234. For information on creating a property list from scratch, see "Creating a saveParamsp script from scratch" on page 255 of the *FDK Programmer's Guide*.

The property list returned in `saveReturnParamspp` has the properties shown in the following table.

Property	Meaning and possible values
`FS_SavedFileName`	A string that specifies the saved file's full pathname.
`FS_SaveNativeError`	The error condition. If the file is saved successfully, it is set to `FE_Success`. See the table below for the possible values.
`FS_SaveStatus`	A bit field indicating what happened when the file was saved. See the table below for the possible values.

Both the `FS_SaveNativeError` property and the `FA_errno` global variable indicate the result of a call to `F_ApiSave()`. The `FS_SaveStatus` flags indicate how or why this result occurred. The following table lists the possible status flags and the `FA_errno` and `FS_SaveNativeError` values associated with them.

FS_SaveNativeError and FA_errno values	Possible FS_SaveStatus flags
`FE_Success`	None.
`FE_Canceled` or `FE_CanceledByClient` (file wasn't saved)	`FV_FileNotWritable`: file was not writable.
	`FV_BadSaveFileName`: specified file name is not allowed by the operating system.
	`FV_BadFileId`: the file's operating system ID was bad.
	`FV_BadSaveScriptValue`: script specified by `saveParamsp` had an invalid value.
	`FV_CancelSaveFileIsInUse`: The file is in use and the user did not or could not reset the lock. Or the file is in use, and the `FS_FileIsInUse` script is set to `FV_DoCancel`, or it is set to `FV_ResetLockAndContinue` but the FrameMaker product could not reset the lock.
	`FV_CancelSaveModDateChanged`: The file has changed since the last time it was opened or saved in the current session.
	`FV_FileWasInUse`: file was in use.
	`FV_LockCouldntBeReset`: file lock couldn't be reset.
	`FV_LockWasReset`: file lock was reset.
	`FV_LockNotReset`: file lock was not reset.

FS_SaveNativeError and FA_errno values	Possible FS_SaveStatus flags
FE_Canceled or FE_CanceledByClient (file wasn't saved)	FV_FileIsViewOnly: file was View Only.
FE_WrongProduct	None. Current FrameMaker product doesn't support this operation
FE_FailedState or FE_BadParameter	None. The filename was invalid.
FE_FilterFailed	FV_InvalidSaveFilter: The filter specified by FS_SaveFileTypeHint is not installed, or the syntax for FS_SaveFileTypeHint is invalid.

After you are done with the property lists you use to call `F_ApiSave()`, use `F_ApiDeallocatePropVals()` to deallocate the memory they use.

Returns

The ID of the document it saved, or `0` if it fails.

Example

See "Example" on page 257 of the *FDK Programmer's Guide* and "F_ApiGetSaveDefaultParams()" on page 234.

See also

"F_ApiDeallocateStructureType()" on page 125, "F_ApiGetSaveDefaultParams()" on page 234, "F_ApiPrintSaveStatus()" on page 372, and "F_ApiSimpleSave()" on page 466.

F_ApiScrollBox()

`F_ApiScrollBox()` displays an array of items and allows the user to choose one.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiScrollBox(IntT *selected_item,
    StringT title,
    F_StringsT *stringslist
    IntT default);
```

Arguments

`selected_item`	The index of the selected item when the user clicks OK (or double-clicks an item). The index of the first item is `0`.
`title`	The title that appears on the dialog box.
`stringslist`	The list of items to appear in the scroll list.
`default`	The index of the item that is selected when the dialog box first appears. For no default, specify `-1`.

IMPORTANT: *If you set* `default` *to* `-1`*, always check to make sure the value returned in* `selected_item` *is* `0` *or greater before you use it as an array index. If you set* `default` *to* `-1` *and the user clicks OK without choosing an item, the value returned in* `selected_item` *will be* `-1`*.*

`F_StringsT` is defined as:

```
typedef struct {
    UIntT len; /* Number of strings */
    StringT *val; /* Array of strings */
} F_StringsT;
```

Returns

`0` if the user clicked OK, or a nonzero value if the user clicked Cancel or an error occurred.

If `F_ApiScrollBox()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	The user clicked Cancel, or a transport error occurred

Example

The following code displays a scroll list dialog box, with a default item labeled “Kurt”:

```
. . .
#include "futils.h"
IntT choice, err;
UCharT msg[256];
F_StringsT names;
StringT nameList[4];

nameList[0] = (StringT)"Kelly";
nameList[1] = (StringT)"Jens";
nameList[2] = (StringT)"Kurt";
nameList[3] = (StringT)"Heycke";

names.len = 4;
names.val = nameList;

err = F_ApiScrollBox(&choice, "Choose a name.", &names, 2);

if (!err)
   F_Sprintf(msg, "The name is %s.", nameList[choice]);
else
   F_Sprintf(msg, "Cancel was pressed");
F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
. . .
```

See also

“F_ApiChooseFile()” on page 97.

F_ApiScrollToText()

`F_ApiScrollToText()` scrolls the document window to a specified text range. It scrolls to the end of the range that is closest to the current display position.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiScrollToText(F_ObjHandleT docId,
   F_TextRangeT *textRangep);
```

Arguments

`docId`	The ID of the document containing the text range
`textRangep`	The text range to which to scroll

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiScrollToText()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_NotTextObject`	One or both of the IDs specified by `textRangep` is not the ID of an object that contains text, such as a paragraph (`FO_Pgf`) or a flow (`FO_Flow`)
`FE_OffsetNotFound`	Offset specified for the text range couldn't be found in the specified paragraph or text line
`FE_BadRange`	Specified text range is invalid

Example

The following code scrolls the document window to the insertion point or the end of the text selection:

```
. . .
F_ObjHandleT docId;
F_TextRangeT tr;

/* Get the insertion point or text selection. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (tr.beg.objId == 0) return;

/* Scroll to it. */
F_ApiScrollToText(docId, &tr);
. . .
```

See also

"F_ApiCenterOnText()" on page 94.

F_ApiService()

`F_ApiService()` services calls from the FrameMaker product to the API. `F_ApiService()` is useful only if you are providing a replacement for `F_ApiRun()`—for example, if your client needs to select on its own file descriptors.

On Windows, F_ApiService parameters are ignored. F_ApiService() waits for and dispatches a single Windows message. If your application contains its own message processing loop you need not call this routine. It is not generally feasible for console applications to receive FDK notifications since there is no way for them to both wait for user imput and call F_ApiService(). Consequently, console applications should not register for notifications.

IMPORTANT: `F_ApiService()` *is only available to Windows RPC clients.*

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiService(IntT *imaskp);
```

Arguments

`imaskp` The address of an integer `select()` mask.

Returns

The number of bits the call to `select()` selected, if any.

If you want other file descriptors to be active at the same time as the API "listen" port, set `imaskp` to the address of an integer `select()` mask that you want to OR into the `F_ApiService()` function's call to `select()`. If the call to `select()` selects on any of the parameter's file descriptors, `F_ApiService()` returns the number of the bits, and puts the bits into `*imaskp`.

Example

The following code sets as the input file descriptor the shell window from which you start the FrameMaker product and processes requests from that file descriptor:

```
. . .
/* Set the shell window as the input fd. */
in_fd = open("/dev/tty",O_RDONLY);

/* Start API. If it fails to start, exit and print error.*/
if (s = F_ApiStartUp(NULL))
   F_ApiErr(s);
else
   while (!FA_bailout)
      {
      myfds = 1<<in_fd;
      F_ApiService(&myfds);
      if (myfds)
         {
         /* . . . run some code . . .*/
         }
      }
. . .
```

Structured **F_ApiSetAttributeDefs()**

`F_ApiSetAttributeDefs()` sets an element definition's attribute definitions.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetAttributeDefs(
    F_ObjHandleT docId,
    F_ObjHandleT elemDefId,
    F_AttributeDefsT *setVal);
```

Arguments

`docId`	The ID of the document containing the element definition
`elemDefId`	The ID of the element definition for which to get attribute definitions
`setVal`	The attribute definitions to set for the element definition

Returns

`VoidT`.

If `F_ApiSetAttributeDefs()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	**Meaning**
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FV_InvAttributeDef`	Invalid attribute name or value
`FE_WrongProduct`	Current FrameMaker product isn't supported

Example

The following code sets the attribute definitions for the Article element definition so that it appears as shown in Figure 2-1:

```
. . .
F_ObjHandleT docId, edefId;
F_AttributeDefsT attributeDefs;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
edefId = F_ApiGetNamedObject(docId, FO_ElementDef, "Article");

attributeDefs.len = 2;
attributeDefs.val = (F_AttributeDefT *)
                    F_Alloc(2*sizeof(F_AttributeDefT), DSE);
attributeDefs.val[0].name = F_StrCopyString("Author");
attributeDefs.val[0].required = True;
attributeDefs.val[0].attrType = FV_AT_STRING;
attributeDefs.val[1].name = F_StrCopyString("Security");
attributeDefs.val[1].required = False;
attributeDefs.val[1].attrType = FV_AT_CHOICES;
attributeDefs.val[1].choices.len = 3;
attributeDefs.val[1].choices.val = (StringT *)
                          F_Alloc(3*sizeof(StringT), DSE);
attributeDefs.val[1].choices.val[0] =
                        F_StrCopyString("Top Secret");
attributeDefs.val[1].choices.val[1] =
                        F_StrCopyString("Classified");
attributeDefs.val[1].choices.val[2] =
                        F_StrCopyString("Unclassified");
F_ApiSetAttributeDefs(docId, edefId, &attributeDefs);
. . .
```

Element (Container): Article
General rule: Para+, Section*
Attribute list

1. **Name:** Author **String** **Required**
2. **Name:** Security **Choice** **Optional**
   Choices: Top Secret, Classified, Unclassified

Figure 2-1 *Article element definition*

See also

"F_ApiGetAttributeDefs()" on page 176.

Structured F_ApiSetAttributes()

`F_ApiSetAttributes()` sets an element's attributes.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetAttributes(
    F_ObjHandleT docId,
    F_ObjHandleT elemId,
    F_AttributesT *setVal);
```

Arguments

`docId`	The ID of the document containing the element
`elemId`	The ID of the element for which to set attributes
`setVal`	The attributes to apply to the specified element

`F_ApiSetAttributes()` sets only the attributes you specify in the `F_AttributesT` structure you pass in `setVal`. If the element has an attribute that you have not specified in the `F_AttributesT` structure, `F_ApiSetAttributes()` does not change the attribute.

If you specify an attribute in the `F_AttributesT` structure that is not defined in the element's element definition, `F_ApiSetAttributes()` adds the attribute to the element as an undefined attribute.

Each attribute in an element is defined by an `F_AttributeT` structure, which is defined as:

```
struct {
    StringT name;
    F_StringsT values;
    UByteT valflags;
    UByteT allow;
} F_AttributeT
```

To delete an attribute value, set `F_AttributeT.values.len = 0` and `F_AttributeT.values.val = NULL`.

Returns

VoidT.

If F_ApiSetAttributes() fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
FE_BadDocId	Invalid document ID
FE_BadObjId	Invalid object ID
FV_InvAttributeDef	Invalid attribute name or value
FE_WrongProduct	Current FrameMaker product isn't supported

Example

The following code sets the Author attribute to jkh and the Security attribute to Classified for the selected element:

```
. . .
F_ObjHandleT docId;
F_AttributesT attributes;
F_ElementRangeT er;
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
er = F_ApiGetElementRange(FV_SessionId,
          docId, FP_ElementSelection);
attributes.len = 2;
attributes.val = (F_AttributeT *)
                    F_Alloc(2*sizeof(F_AttributeT), DSE);
attributes.val[0].name = F_StrCopyString("Author");
attributes.val[0].values.len = 1;
attributes.val[0].values.val = (StringT *)
                           F_Alloc(sizeof(StringT), DSE);
attributes.val[0].values.val[0] = F_StrCopyString("jkh");
attributes.val[1].name = F_StrCopyString("Security");
attributes.val[1].values.len = 1;
attributes.val[1].values.val = (StringT *)
                           F_Alloc(sizeof(StringT), DSE);
attributes.val[1].values.val[0] = F_StrCopyString("Classified");
F_ApiSetAttributes(docId, er.beg.childId, &attributes);
. . .
```

See also

"F_ApiGetAttributes()" on page 178.

F_ApiSetAttributeSimple()

`F_ApiSetAttributeSimple()` sets attribute value for element(s).

Synopsis

```
#include "fapi.h"
. . .
StatusT F_ApiSetAttributeSimple(F_ObjHandleT docId, ConStringT
attrName, ConStringT attrValStr, IntT scope, F_ObjHandleT
objId);
```

Arguments

`docId`	The ID of the document.
`attrName`	The name of the attribute which has to be set or changed
`attrValStr`	The attribute value that is to be set. If this is specified as `null`, the attribute will be deleted from the element.
`scope`	One of `FV_Element`, `FV_ElementsOfType` or `FV_AllElements`.
`objId`	When `scope` is `FV_Element`, objId should be the ID of the element object on which attribute is to be set. When `scope` is `FV_ElementsOfType`, objId should be the ID of the element definition object. Attribute value will be set on all occurrences of elements with this element definition. When `scope` is `FV_AllElements`, objId has no effect. Attribute value will be set on all elements.

Returns

Returns `FE_Success` if the operation is successful. Else sets `FA_errno` to one of the following:

- `FE_WrongProduct`: The product is not Structured FrameMaker.
- `FE_BadDocId`: Bad document id.
- `FE_BadElementId`: Bad element id when scope is `FV_Element`.
- `FE_BadElementDefId`: Bad element definition id when scope is `FV_ElementsOfType`
- `FE_BadParameter`: Scope is not one of the specified values

F_ApiSetClientDir()

`F_ApiSetClientDir()` sets the default directory for client operations, such as opening resources. It overrides the default directory setting.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetClientDir(StringT dirName);
```

Arguments

`dirName` The name of the directory to set as the default directory for client operations.

Returns

`VoidT`.

Example

The following code sets the client's default directory to `/tmp`:

```
. . .
F_ApiSetClientDir("/tmp");
. . .
```

See also

"F_ApiClientDir()" on page 103, "F_ApiClientName()" on page 104, "F_ApiOpenResource()" on page 363.

Structured F_ApiSetElementRange()

`F_ApiSetElementRange()` sets an element range (`F_ElementRangeT`) property.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetElementRange(
    F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    F_ElementRangeT *setVal);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to set. If the object is a session, specify `FV_SessionId`.
`objId`	The ID of the object whose property you want to set.
`propNum`	The property to set. Specify an API-defined constant, such as `FP_ElementSelection`.
`setVal`	The element range to set the property to.

The `F_ElementRangeT` structure specifies an element range. The `F_ElementRangeT` structure is defined as:

```
typedef struct {
   F_ElementLocT beg; /* Beginning of the element range. */
   F_ElementLocT end; /* End of the element range. */
} F_ElementRangeT;
```

The `F_ElementLocT` structure specifies a location within an element. It is defined as:

```
typedef struct {
   F_ObjHandleT parentId; /* Parent element ID. */
   F_ObjHandleT childId; /* Child element ID. */
   IntT offset; /* Offset within child/parent element. */
} F_ElementLocT;
```

To specify a selection that includes the root element, set `beg.parentId` to `0`, `beg.childId` to the ID of the root element, and `end.childId` to `0`.

Returns

`VoidT`.

If `F_ApiSetElementRange()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	One or more invalid object ids were specified in the `F_ElementRangeT` that was passed in the `setVal` argument
`FE_WrongProduct`	Current FrameMaker product isn't supported

Example

The following code selects the entire parent element of the selected element (or the element containing the text selection):

```
. . .
F_ObjHandleT docId, parentId;
F_ElementRangeT er;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
er = F_ApiGetElementRange(FV_SessionId,
          docId, FP_ElementSelection);

parentId = F_ApiGetId(docId, er.beg.childId, FP_ParentElement);
er.end.parentId = er.beg.parentId = F_ApiGetId(docId, parentId,
                        FP_ParentElement);

/* If the selected element is a child of the highest level
** element, the client returns here.
*/
if(!er.end.parentId) return;
er.beg.childId = parentId;
er.beg.offset = er.end.offset = 0;
er.end.childId = F_ApiGetId(docId, parentId,
                            FP_NextSiblingElement);
F_ApiSetElementRange(0, docId, FP_ElementSelection, &er);
. . .
```

See also

"F_ApiGetElementRange()" on page 183.

F_ApiSetId()

`F_ApiSetId()` sets an ID property.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetId(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    F_ObjHandleT setVal);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to set. If the object is a session, specify `0`.
`objId`	The ID of the object whose property you want to set.
`propNum`	The property to set (for example, `FP_NextGraphicInFrame`).
`setVal`	The value to which to set the property.

Returns

`VoidT`.

If `F_ApiSetId()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadNewFrame`	The API can't move the specified object to this frame
`FE_BadNewGroup`	The API can't move the specified object to this graphic object group (`FO_Group`)
`FE_BadNewSibling`	Object can't be made a sibling of the specified object
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_BadRange`	Specified text range is invalid
`FE_GroupSelect`	The API can't select or deselect an object in the specified group
`FE_HiddenPage`	Value must be the ID of a hidden page (`FO_HiddenPage`)
`FE_NotBookComponent`	Value must be the ID of a book component (`FO_BookComponent`)
`FE_NotFrame`	Value must be the ID of a frame
`FE_NotGraphic`	Value must be the ID of a graphic object
`FE_NotGroup`	Value must be the ID of a graphic object group (`FO_Group`)
`FE_NotInMenu`	Value must be the ID of a command (`FO_Command`) or a menu (`FO_Menu`) in the specified menu
`FE_NotMenu`	Object must be a menu (`FO_Menu`)

FA_errno value	Meaning
FE_NotTextFrame	Value must be the ID of a text column (FO_TextFrame)
FE_NotTextObject	Object must be an object that contains text, such as a paragraph (FO_Pgf) or a flow (FO_Flow)
FE_OutOfRange	Specified property value is out of the legal range for the specified property
FE_PageFrame	Value must be the ID of a page frame object (FO_UnanchoredFrame)
FE_ReadOnly	Property is read-only
FE_WithinFrame	Object must be moved to a different frame first
FE_WrongProduct	Current FrameMaker product doesn't support the operation

Example

The following code moves the graphic objects in the selected frame so that they appear directly on the underlying page. It does this by making the page frame the objects' parent. For information on page frames and how the API organizes graphics, see "How the API represents pages" on page 86 and "How the API organizes graphic objects" on page 92 of the *FDK Programmer's Guide*:

```
. . .
F_ObjHandleT frameId, pFrameId, pgId, docId, objId, nextObjId;

frameId = 0;

/* Get ID of active document and selected frame. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
frameId = F_ApiGetId(FV_SessionId, docId,
                        FP_FirstSelectedGraphicInDoc);

if (!frameId) return; /* No frame has been selected. */

/* Get ID of current page and its page frame. */
pgId = F_ApiGetId(FV_SessionId, docId, FP_CurrentPage);
pFrameId = F_ApiGetId(docId, pgId, FP_PageFrame);

/* Move all the objects to current page's page frame. */
objId = F_ApiGetId(docId, frameId, FP_FirstGraphicInFrame);
while(objId)
    {
    /* Change the object's parent to be the page frame. */
    nextObjId = F_ApiGetId(docId, objId, FP_NextGraphicInFrame);
    F_ApiSetId(docId, objId, FP_FrameParent, pFrameId);
    objId = nextObjId;
    }
. . .
```

See also

"F_ApiGetId()" on page 191.

F_ApiSetInt()

`F_ApiSetInt()` sets an integer property. Integer properties include ordinal, `True/False` (Boolean), and enumerated properties.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetInt(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    IntT setVal);
```

Arguments

`docId`	The ID of the document, dialog box, book, or session containing the object whose property you want to set. If the object is a session, specify `0`.
`objId`	The ID of the object whose property you want to set.
`propNum`	The property to set (for example, `FP_ShowAll`).
`setVal`	The value to which to set the property.

Returns

`VoidT`.

If `F_ApiSetInt()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_CantSmooth`	Object can’t be smoothed
`FE_NotApiCommand`	Object must be the ID of a command defined by an FDK client
`FE_OutOfRange`	Specified property value is out of the legal range for the specified property
`FE_ReadOnly`	Property is read-only
`FE_WrongProduct`	Current FrameMaker product doesn’t support the operation

Example

The following code turns off automatic save for a session:

```
. . .
F_ApiSetInt(0, FV_SessionId, FP_AutoSave, False);
. . .
```

See also

"F_ApiGetInt()" on page 201.

F_ApiSetIntByName()

`F_ApiSetIntByName()` sets an integer (`IntT`) inset facet.

`F_ApiSetIntByName()` and other `F_ApiSet`*`PropertyType`*`ByName()` functions use a transaction model to set facets. After you have finished setting facets, you must commit the transaction by using `F_ApiSetIntByName()` to set a facet named `""` to `0`.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetIntByName(F_ObjHandleT docId,
       F_ObjHandleT objId,
       StringT propName,
       IntT setVal);
```

Arguments

`docId`	The ID of the document containing the inset whose facet you want to set
`objId`	The ID of the inset whose facet you want to set
`propName`	The name of the facet to set
`setVal`	The value to which to set the facet

Returns

`VoidT`.

If `F_ApiSetIntByName()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadName`	Specified name is illegal
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code sets an integer facet named `revision.facet` to `4`:

```
. . .
F_ObjHandleT docId, insetId;

F_ApiSetIntByName(docId, insetId, "revision.facet", 4);
F_ApiSetIntByName(docId, insetId, "", 0); /* Commit. */
. . .
```

See also

"F_ApiGetIntByName()" on page 203.

F_ApiSetInts()

`F_ApiSetInts()` sets an `F_IntsT` property.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetInts(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    F_IntsT *setVal);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to set
`objId`	The ID of the object whose property you want to set
`propNum`	The property to set (for example, `FP_InCond`)
`setVal`	The `F_IntsT` structure to which to set the property

`F_IntsT` is defined as:

```
typedef struct {
    UIntT len; /* Number of IntTs */
    IntT *val; /* Array of IntTs or F_ObjHandleTs */
} F_IntsT;
```

Returns

`VoidT`.

If `F_ApiSetInts()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_OutOfRange`	Specified property value is out of the legal range for the specified property

FA_errno value	Meaning
FE_ReadOnly	Property is read-only
FE_WrongProduct	Current FrameMaker product doesn't support the operation

Example

The following code sets the condition for text added at the insertion point to Comment:

```
. . .
F_ObjHandleT docId, commentId;
F_IntsT conditionIds;

/* Get ID of Comment tag in current document. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
commentId = F_ApiGetNamedObject(docId, FO_CondFmt, "Comment");

conditionIds.val = (IntT*) &commentId;
conditionIds.len = 1;

/* Set document's type-in condition property to Comment ID. */
F_ApiSetInts(FV_SessionId, docId, FP_InCond, &conditionIds);
. . .
```

See also

"F_ApiGetInts()" on page 204.

F_ApiSetMetric()

`F_ApiSetMetric()` sets a metric (`MetricT`) property.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetMetric(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    MetricT setVal);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to set
`objId`	The ID of the object whose property you want to set
`propNum`	The property to set (for example, `FP_Leading`)
`setVal`	The metric value to which to set the property

Returns

`VoidT`.

If `F_ApiSetMetric()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_OutOfRange`	Specified property value is out of the legal range for the specified property
`FE_ReadOnly`	Property is read-only
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code sets the zoom factor for all the documents in the current book to 200 percent:

```
. . .
#include "fmemory.h"
F_ObjHandleT bookId, compId, docId;
StringT docName;

bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook);
compId = F_ApiGetId(bookId, bookId, FP_FirstComponentInBook);

while(compId) /* Traverse book components and open documents. */
    {
    /* Get the document's name and open it. */
    docName = F_ApiGetString(bookId, compId, FP_Name);
    docId = F_ApiSimpleOpen(docName, False);

    /* Set zoom. 100% = MetricT 1<<16 or 65536. */
    F_ApiSetMetric(FV_SessionId, docId, FP_Zoom, 2*65536);

    /* Save and close the document. */
    F_ApiSimpleSave(docId, docName, False);
    F_ApiDeallocateString(&docName);
    F_ApiClose(docId, True);

    compId = F_ApiGetId(bookId, compId, FP_NextComponentInBook);
    }
. . .
```

For this code to work correctly, all the documents in the book must initially be closed.

See also

"F_ApiGetMetric()" on page 209.

F_ApiSetMetricByName()

`F_ApiSetMetricByName()` sets a metric (`MetricT`) inset facet.

`F_ApiSetMetricByName()` and other `F_ApiSetByName()` functions use a transaction model to set facets. After you have finished setting facets, you must commit the transaction by using `F_ApiSetIntByName()` to set a facet named `""` to `0`.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetMetricByName(F_ObjHandleT docId,
       F_ObjHandleT objId,
       StringT propName,
       MetricT setVal);
```

Arguments

`docId`	The ID of the document containing the inset
`objId`	The ID of the inset whose facet you want to set
`propName`	The name of the facet to set
`setVal`	The value to which to set the facet

Returns

`VoidT`.

If `F_ApiSetMetricByName()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadObjId`	Invalid object ID
`FE_BadPropType`	Incorrect property type for this function
`FE_OutOfRange`	Specified property value is out of the legal range for the specified property
`FE_ReadOnly`	Property is read-only
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code sets an integer facet named `height.facet` to 4 inches:

```
. . .
#define in ((MetricT) 65536*72)
F_ObjHandleT docId, insetId;

F_ApiSetMetricByName(docId, insetId, "height.facet", 4*in);
F_ApiSetIntByName(docId, insetId, "", 0); /* Commit. */
. . .
```

See also

"F_ApiGetMetricByName()" on page 210.

F_ApiSetMetrics()

`F_ApiSetMetrics()` sets a metrics (`F_MetricsT`) property.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetMetrics(F_ObjHandleT docId,
       F_ObjHandleT objId,
       IntT propNum,
       F_MetricsT *setVal);
```

Arguments

`docId`	The ID of the document, book, or session containing the object whose property you want to set
`objId`	The ID of the object whose property you want to set
`propNum`	The property to query (for example, `FP_TblColWidths`)
`setVal`	The `F_MetricsT` structure to which to set the property

`F_MetricsT` is defined as:

```
typedef struct {
   UIntT len; /* Number of metric values*/
   MetricT *val; /* Array of metric values */
} F_MetricsT;
```

Returns

`VoidT`.

If `F_ApiSetMetrics()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_OutOfRange`	Specified property value is out of the valid range for the specified property
`FE_ReadOnly`	Property is read-only
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code sets the line pattern of a graphic object to a 14-point dash/1-point space dashed pattern:

```
. . .
#include "fmemory.h"
#define pts (MetricT) 65536

F_MetricsT dashPat;
F_ObjHandleT docId, objId;
MetricT vals[2];

/* Get IDs of active document and selected object. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
objId = F_ApiGetId(FV_SessionId, docId,
                   FP_FirstSelectedGraphicInDoc);

/* Create array containing dash pattern. */
dashPat.len = 2;
dashPat.val = vals;
dashPat.val[0] = 14*pts;         /* 14-pt dash */
dashPat.val[1] = pts;     /* 1-pt space */

/* Set selected object's dash pattern. */
F_ApiSetMetrics(docId, objId, FP_Dash, &dashPat);
. . .
```

See also

"F_ApiGetMetrics()" on page 212.

F_ApiSetPoints()

`F_ApiSetPoints()` sets the array of points (or vertices) for a polygon or polyline object.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetPoints(F_ObjHandleT docId,
   F_ObjHandleT objId,
   IntT propNum,
   F_PointsT *setVal);
```

Arguments

`docId`	The ID of the document containing the object whose property you want to set
`objId`	The ID of the object whose property you want to set
`propNum`	The property to set (for example, `FP_Points`)
`setVal`	The `F_PointsT` structure to which to set the property

The `F_PointsT` structure is defined as:

```
typedef struct {
   UIntT len; /* Number of coordinate pairs */
   F_PointT *val; /* Vector of coordinate pairs */
} F_PointsT;
```

The `F_PointT` structure, which specifies an individual x-y coordinate pair, is defined as:

```
typedef struct{
   MetricT    x, y; /* The coordinates */
} F_PointT;
```

Returns

`VoidT`.

If `F_ApiSetPoints()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_OutOfRange`	Specified property value is out of the legal range for the specified property
`FE_ReadOnly`	Property is read-only
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code creates a triangle on the current page's page frame:

```
. . .
#define pts (MetricT) 65536
F_ObjHandleT docId, pageId, pFrameId, polyId;
F_PointsT points;
F_PointT vals[3];

/* Get IDs of active document, current page, and page frame. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
pageId = F_ApiGetId(docId, docId, FP_CurrentPage);
pFrameId = F_ApiGetId(docId, pageId, FP_PageFrame);

/* Set up array of triangle's vertices. */
points.val = vals;
points.len = 3;
points.val[0].x = 60*pts;
points.val[0].y = 20*pts;
points.val[1].x = 20*pts;
points.val[1].y = 80*pts;
points.val[2].x = 80*pts;
points.val[2].y = 80*pts;

/* Create the polygon on page frame, then set its vertices. */
polyId = F_ApiNewGraphicObject(docId, FO_Polygon, pFrameId);
F_ApiSetPoints(docId, polyId, FP_Points, &points);
. . .
```

See also

"F_ApiGetPoints()" on page 228.

F_ApiSetProps()

`F_ApiSetProps()` applies a property list to a specified object.

You can use `F_ApiSetProps()` and `F_ApiGetProps()` to copy formats from one object to another. For example, you can copy properties from one graphic object to another, from a paragraph format to a paragraph, or from one paragraph to another paragraph.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetProps(F_ObjHandleT docId,
    F_ObjHandleT objId,
    F_PropValsT *setVal);
```

Arguments

`docId`	The ID of the dialog box or document containing the object
`objId`	The ID of the object that you want to apply the property list to
`setVal`	The property list

Returns

`VoidT`.

If `F_ApiSetProps()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadNewFrame`	The API can't move the specified object to this frame
`FE_BadNewGroup`	The API can't move the specified object to this graphic object group (`FO_Group`)
`FE_BadNewSibling`	Object can't be made a sibling of the specified object
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_BadRange`	Specified text range is invalid

FA_errno value	Meaning
FE_CantSmooth	Object can't be smoothed
FE_DupName	Property can't be set to this name because it is already used by another object
FE_GenRuleAmbiguous	General rule in structured document was ambiguous
FE_GenRule ConnectorExpected	General rule in structured document missing connector
FE_GenRuleItem Expected	General rule in structured document missing rule item
FE_GenRuleLeft BracketExpected	General rule in structured document missing left bracket
FE_GenRuleMixed Connectors	General rule in structured document has mixed connectors
FE_GenRuleRight BracketExpected	General rule in structured document missing right bracket
FE_GenRuleSyntax Error	General rule in structured document has syntax error
FE_GroupSelect	The API can't select or deselect an object in the specified group
FE_HiddenPage	Value must be the ID of a hidden page (FO_HiddenPage)
FE_InvContextSpec	The API encountered an invalid context specification in a Structured FrameMaker document
FE_NotBookComponent	Value must be the ID of a book component (FO_BookComponent)
FE_NotFrame	Value must be the ID of a frame
FE_NotGraphic	Value must be the ID of a graphic object
FE_NotGroup	Value must be the ID of a graphic object group (FO_Group)
FE_NotTextFrame	Value must be the ID of a text column (FO_TextFrame)
FE_NotTextObject	Object must be an object that contains text, such as a paragraph (FO_Pgf) or a flow (FO_Flow)
FE_OffsetNotFound	Offset specified for the text location couldn't be found in the specified paragraph or text line
FE_OutOfRange	Specified property value is out of the legal range for the specified property
FE_PageFrame	Value must be the ID of a page frame object (FO_UnanchoredFrame)

FA_errno value	Meaning
FE_ReadOnly	Property is read-only
FE_WithinFrame	Object must be moved to a different frame first
FE_WrongProduct	Current FrameMaker product doesn't support the operation

Example

See "Getting and setting property lists" on page 291 of the *FDK Programmer's Guide*.

F_ApiSetPropVal()

F_ApiSetPropVal() sets a property of any type. If you know a property's type, it is normally easier to call an F_ApiSet() function, such as F_ApiSetInt(), to set it.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetPropVal(F_ObjHandleT docId,
    F_ObjHandleT objId,
    F_PropValT *setVal);
```

Arguments

docId	The ID of the document, book, or session containing the object whose property you want to set. If the object is a session, specify 0.
objId	The ID of the object whose property you want to set.
setVal	The property to set for the specified object.

Returns

VoidT.

If `F_ApiSetPropVal()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadNewFrame`	The API can't move the specified object to this frame
`FE_BadNewGroup`	The API can't move the specified object to this graphic object group (`FO_Group`)
`FE_BadNewSibling`	Object can't be made a sibling of the specified object
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_BadRange`	Specified text range is invalid
`FE_GroupSelect`	The API can't select or deselect an object in the specified group
`FE_HiddenPage`	Value must be the ID of a hidden page (`FO_HiddenPage`)
`FE_NotBookComponent`	Value must be the ID of a book component (`FO_BookComponent`)
`FE_NotFrame`	Value must be the ID of a frame
`FE_NotGraphic`	Value must be the ID of a graphic object
`FE_NotGroup`	Value must be the ID of a graphic object group (`FO_Group`)
`FE_NotTextFrame`	Value must be the ID of a text column (`FO_TextFrame`)
`FE_NotTextObject`	Object must be an object that contains text, such as a paragraph (`FO_Pgf`) or a flow (`FO_Flow`)
`FE_PageFrame`	Value must be the ID of a page frame object (`FO_UnanchoredFrame`)
`FE_ReadOnly`	Property is read-only
`FE_WithinFrame`	Object must be moved to a different frame first
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code turns off the automatic save feature:

```
. . .
F_PropValT setVal;

setVal.propIdent.num = FP_AutoSave;
setVal.propVal.valType = FT_Integer;
setVal.propVal.u.ival = False;
F_ApiSetPropVal(0, FV_SessionId, &setVal);
. . .
```

See also

"F_ApiGetProps()" on page 231 and "F_ApiSetProps()" on page 432.

F_ApiSetString()

`F_ApiSetString()` sets a string (`StringT`) property.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetString(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    StringT setVal);
```

Arguments

`docId`	The ID of the document, dialog box, book, or session containing the object whose property you want to set. If the object is a session, specify `0`.
`objId`	The ID of the object whose property you want to set.
`propNum`	The property to set (for example, `FP_MarkerText`).
`setVal`	The string to set the property to.

Returns

`VoidT`.

If `F_ApiSetString()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_BadRange`	Specified text range is invalid
`FE_DupName`	Property can't be set to this name because it is already used by another object
`FE_GenRuleAmbiguous`	General rule in structured document was ambiguous
`FE_GenRuleConnector Expected`	General rule in structured document missing connector
`FE_GenRuleItemExpected`	General rule in structured document missing rule item
`FE_GenRuleLeftBracket Expected`	General rule in structured document missing left bracket
`FE_GenRuleMixed Connectors`	General rule in structured document has mixed connectors
`FE_GenRuleRightBracket Expected`	General rule in structured document missing right bracket
`FE_GenRuleSyntaxError`	General rule in structured document has syntax error
`FE_NotApiCommand`	Object must be the ID of a command defined by an FDK client
`FE_OutOfRange`	Specified property value is out of the legal range for the specified property
`FE_ReadOnly`	Property is read-only
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code sets the printer to `ps11` for the active document:

```
. . .
F_ObjHandleT docId;
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
F_ApiSetString(FV_SessionId, docId, FP_PrinterName, "ps11");
. . .
```

See also

"F_ApiGetString()" on page 241.

F_ApiSetStrings()

`F_ApiSetStrings()` sets a strings (`F_StringsT`) property. For example, you can use it to add to the list of words in a document's dictionary.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetStrings(F_ObjHandleT docId,
   F_ObjHandleT objId,
   IntT propNum,
   F_StringsT *setVal);
```

Arguments

`docId`	The ID of the document, dialog box, book, or session containing the object whose property you want to set. For a session, specify `0`.
`objId`	The ID of the object whose property you want to set.
`propNum`	The property to set (for example, `FP_Dictionary`).
`setVal`	The `F_StringsT` structure to which to set the property.

The `F_StringsT` structure is defined as:

```
typedef struct {
   UIntT len; /* The number of strings */
   StringT *val; /* The array of strings */
} F_StringsT;
```

Returns

`VoidT`.

If `F_ApiSetStrings()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal; when setting `FP_PDFDocInfo`, there was an entry name that contained non-printable ASCII, an invalid Hex code, or the entry name was an empty string; see "PDF Document Info dictionaries" on page 729
`FE_BadObjId`	Invalid object ID
`FE_BadParameter`	When setting `F_StringsT` for `FP_PDFDocInfo`, there was an odd number of strings; see "PDF Document Info dictionaries" on page 729
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_BadShortCut`	Specified shortcut is not valid for the current platform
`FE_InvalidString`	When setting `FP_PDFDocInfo`, one of the submitted characters does not translate to valid Unicode, or the string is too long; see "PDF Document Info dictionaries" on page 729
`FE_ReadOnly`	Property is read-only
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code prompts the user to enter a word, then adds the word to the active document's dictionary:

```
. . .
#include "fmemory.h"
StringT sres;
F_ObjHandleT docId;
F_StringsT strings;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Prompt user for a word to add. Return if Cancel clicked. */
if (F_ApiPromptString(&sres, "Enter word to add to dictionary",
                    "") != 0) return;

/* Get the document dictionary word list. */
strings = F_ApiGetStrings(FV_SessionId, docId, FP_Dictionary);

/* Allocate space for the new word. */
if (strings.len++)
   strings.val = (StringT *) F_Realloc(strings.val,
             strings.len*sizeof(StringT), NO_DSE);
else
    strings.val = (StringT*) F_Alloc(sizeof(StringT), NO_DSE);

/* Add word to the dictionary words. */
strings.val[strings.len-1] = sres;

/* Set property to the amended word list. */
F_ApiSetStrings(FV_SessionId, docId, FP_Dictionary, &strings);
. . .
```

See also

"F_ApiGetStrings()" on page 242.

F_ApiSetTabs()

`F_ApiSetTabs()` sets the array of tabs for a paragraph or paragraph format. It automatically orders the tabs that you pass to it.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetTabs(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    F_TabsT* setVal);
```

Arguments

`docId`	The ID of the document containing the object whose property you want to set
`objId`	The ID of the object whose property you want to set
`propNum`	The property to set (for example, `FP_Tabs`)
`setVal`	The `F_TabsT` structure to which to set the property

You don't need to insert tabs in order into the `setVal` array; you can just append them. The API will sort them for you.

The `F_TabsT` structure is defined as:

```
typedef struct {
    UIntT len; /* The number of tabs in val */
    F_TabT *val; /* Structures that describe the tabs */
} F_TabsT;
```

The `F_TabT` structure is defined as:

```
typedef struct {
    MetricT x; /* Offset from paragraph's left margin */
    UCharT type; /* Constant for tab type, e.g. FV_TAB_RIGHT */
    StringT leader; /* Characters before tab, e.g. "." */
    UCharT decimal; /* Decimal tab character, e.g. '.' */
} F_TabT;
```

`F_TabT.type` can be one of the following.

Type constant	Tab type
`FV_TAB_LEFT`	Left tab
`FV_TAB_CENTER`	Center tab
`FV_TAB_RIGHT`	Right tab
`FV_TAB_DECIMAL`	Decimal tab
`FV_TAB_RELATIVE_LEFT`	Relative center tab (allowed only for format change lists)
`FV_TAB_RELATIVE_CENTER`	Relative right tab (allowed only for format change lists)
`FV_TAB_RELATIVE_RIGHT`	Relative decimal tab (allowed only for format change lists)
`FV_TAB_RELATIVE_DECIMAL`	Relative center tab (allowed only for format change lists)

IMPORTANT: *If you set an erronious value for a tab type in any* `F_TabT`*, this function will not set any tabs, and it will return* `FE_Success`*. You should also be careful not to set a tab to a location that is off the page.*

Returns

`VoidT`.

If `F_ApiSetTabs()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_OutOfRange`	Specified property value is out of the legal range for the specified property
`FE_ReadOnly`	Property is read-only
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code adds a 2.0-inch decimal tab in the paragraph containing the insertion point or the beginning of the current text selection.

```
. . .
#include "fstrings.h"
#include "fmemory.h"
#define in ((MetricT) 65536*72)

F_ObjHandleT docId;
F_TabsT tabs;
F_TextRangeT tr;

/* Get insertion point or current text selection. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (tr.beg.objId == 0) return;

/* Get the tabs and allocate space for new tab. */
tabs = F_ApiGetTabs(docId, tr.beg.objId, FP_Tabs);
if (tabs.len++)
   tabs.val=(F_TabT*)F_Realloc(tabs.val,
      tabs.len*sizeof(F_TabT), NO_DSE);
else
   tabs.val = (F_TabT*) F_Alloc(sizeof(F_TabT),NO_DSE);

/* Add the tab to the array. */
tabs.val[tabs.len-1].type = FV_TAB_DECIMAL;
tabs.val[tabs.len-1].x = 2*in;
tabs.val[tabs.len-1].decimal = '.';
tabs.val[tabs.len-1].leader = (StringT)F_StrCopyString(" ");

/* Set the FP_Tabs property to the new set of tabs. */
F_ApiSetTabs(docId, tr.beg.objId, FP_Tabs, &tabs);
F_ApiDeallocateTabs(&tabs);
. . .
```

See also

“F_ApiGetTabs()” on page 245.

F_ApiSetTextLoc()

`F_ApiSetTextLoc()` sets a text location (`F_TextLocT`) property.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetTextLoc(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    F_TextLocT *setval);
```

Arguments

`docId`	The ID of the document containing the object whose property you want to set.
`objId`	The ID of the object whose property you want to set.
`propNum`	The property to set.
`setVal`	The text location to which to set the property. For information on specifying text locations, see "Getting and setting the insertion point or text selection" on page 321 of the *FDK Programmer's Guide*.

The `F_TextLocT` structure is defined as:

```
typedef struct{
    F_ObjHandleT objId; /* Id of object containing text. */
    IntT offset; /* From start of object. */
} F_TextLocT;
```

Returns

`VoidT`.

If `F_ApiSetTextLoc()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadObjId`	Invalid object ID
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified text object

FA_errno value	Meaning
FE_OutOfRange	Specified property value is out of the legal range for the specified property
FE_BadPropNum	Specified property number is invalid
FE_BadPropType	Incorrect property type for this function
FE_ReadOnly	Property is read-only
FE_WrongProduct	Current FrameMaker product doesn't support the operation

See also

"F_ApiGetTextLoc()" on page 264.

F_ApiSetTextProps()

F_ApiSetTextProps() sets the text properties (such as the format tag, font family, and size) for a text range.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetTextProps(F_ObjHandleT docId,
    F_TextRangeT *textRangep,
    F_PropValsT *setVal);
```

Arguments

docId	The ID of the document containing the text whose property list you want to set.
textRangep	The text range to which to apply the property list. For information on specifying text ranges, see "Getting and setting the insertion point or text selection" on page 321 of the *FDK Programmer's Guide*.
setVal	The property list to apply to the text range.

Returns

VoidT.

If `F_ApiSetTextProps()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadRange`	Specified text range is invalid
`FE_GenRuleAmbiguous`	General rule in structured document was ambiguous
`FE_GenRuleConnector Expected`	General rule in structured document missing connector
`FE_GenRuleItemExpected`	General rule in structured document missing rule item
`FE_GenRuleLeftBracket Expected`	General rule in structured document missing left bracket
`FE_GenRuleMixed Connectors`	General rule in structured document has mixed connectors
`FE_GenRuleRightBracket Expected`	General rule in structured document missing right bracket
`FE_GenRuleSyntaxError`	General rule in structured document has syntax error
`FE_NotTextObject`	Value must be the ID of an object that contains text, such as a paragraph (`FO_Pgf`) or a flow (`FO_Flow`)
`FE_OffsetNotFound`	Offset specified for the text location couldn't be found in the specified paragraph or text line
`FE_OutOfRange`	Specified property value is out of the legal range for the specified property
`FE_ReadOnly`	Property is read-only
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

Example

The following code sets the size of the selected text to 30 points and sets the underlining to a single underline:

```
. . .
#define pts (MetricT) 65536 /* A Frame metric point. */
F_TextRangeT tr;
F_PropValsT props;
F_ObjHandleT docId;

/* Get current text selection. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (tr.beg.objId == 0) return;

/* Allocate memory for the list. */
props = F_ApiAllocatePropVals(2);

/* Set up the properties. */
props.val[0].propIdent.num = FP_FontSize;
props.val[0].propVal.valType = FT_Metric;
props.val[0].propVal.u.ival = 30 * pts;

props.val[1].propIdent.num = FP_Underlining;
props.val[1].propVal.valType = FT_Integer;
props.val[1].propVal.u.ival = FV_CS_SINGLE_UNDERLINE;

/* Apply the property list to the text selection. */
F_ApiSetTextProps(docId, &tr, &props);

/* Deallocate memory referenced by the property list. */
F_ApiDeallocatePropVals(&props);
. . .
```

See also

"F_ApiGetTextProps()" on page 265.

F_ApiSetTextPropVal()

`F_ApiSetTextPropVal()` sets a text property for a specified text range. The property can be of any type.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetTextPropVal(F_ObjHandleT docId,
    F_TextRangeT *textRangep,
    F_PropValT *setVal);
```

Arguments

`docId`	The ID of the document containing the text range.
`textRangep`	The text range to apply the property to. For information on specifying text locations, see "Getting and setting the insertion point or text selection" on page 321 of the *FDK Programmer's Guide*.
`setVal`	The property to apply to the text range.

Returns

`VoidT`.

If `F_ApiSetTextPropVal()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_BadRange`	Specified text range is invalid
`FE_GenRuleAmbiguous`	General rule in structured document was ambiguous
`FE_GenRuleConnector Expected`	General rule in structured document missing connector
`FE_GenRuleItemExpected`	General rule in structured document missing rule item
`FE_GenRuleLeftBracket Expected`	General rule in structured document missing left bracket
`FE_GenRuleMixed Connectors`	General rule in structured document has mixed connectors
`FE_GenRuleRightBracket Expected`	General rule in structured document missing right bracket

FA_errno value	Meaning
FE_GenRuleSyntaxError	General rule in structured document has syntax error
FE_NotTextObject	Value must be the ID of an object that contains text, such as a paragraph (FO_Pgf) or a flow (FO_Flow)
FE_OffsetNotFound	Offset specified for the text location couldn't be found in the specified paragraph or text line
FE_OutOfRange	Specified property value is out of the legal range for the specified property
FE_ReadOnly	Property is read-only
FE_WrongProduct	Current FrameMaker product doesn't support the operation

Example

The following code turns off change bars for the selected text:

```
. . .
F_TextRangeT tr;
F_PropValT prop;
F_ObjHandleT docId;

/* Get current text selection. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (tr.beg.objId == 0) return;

/* Set up the property. */
prop.propIdent.num = FP_ChangeBar;
prop.propVal.valType = FT_Integer;
prop.propVal.u.ival = False;

/* Apply the property to the text selection. */
F_ApiSetTextPropVal(docId, &tr, &prop);
. . .
```

See also

"F_ApiGetTextPropVal()" on page 267, "F_ApiSetTextProps()" on page 445, and "F_ApiSetTextVal()" on page 452.

F_ApiSetTextRange()

`F_ApiSetTextRange()` sets a text range (`F_TextRangeT`) property.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetTextRange(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    F_TextRangeT *setVal);
```

Arguments

`docId`	The ID of the document containing the object whose property you want to set.
`objId`	The ID of the object whose property you want to set.
`propNum`	The property to set.
`setVal`	The text range to which to set the "Getting and setting the insertion point or text selection" on page 321 of the *FDK Programmer's Guide*.

Returns

`VoidT`.

If `F_ApiSetTextRange()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadObjId`	Invalid object ID
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_OutOfRange`	Specified property value is out of the legal range for the specified property
`FE_ReadOnly`	Property is read-only
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation

FA_errno value	Meaning
FE_NotTextObject	Specified text range includes a table row; for example text locations before and after a structured table row element
FE_BadRange	Specified text range includes an entire table cell; for example locations before and after a structured table cell element

Example

The following code gets the insertion point or text selection, and then extends the selection from the insertion point or text selection to the beginning of the next paragraph:

```
. . .
F_TextRangeT tr;
F_ObjHandleT docId, nextpgfId;

/* Get ID of active document. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Get the current text selection. */
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (tr.beg.objId == 0) return;

/* Get ID of the next paragraph and extend selection to it. */
nextpgfId = F_ApiGetId(docId, tr.end.objId, FP_NextPgfInFlow);
if (nextpgfId){
   tr.end.objId = nextpgfId;
   tr.end.offset = 0;
   F_ApiSetTextRange(FV_SessionId, docId, FP_TextSelection,&tr);
}
. . .
```

See also

“F_ApiGetTextRange()” on page 269.

F_ApiSetTextVal()

`F_ApiSetTextVal()` sets a specified text property for a text range.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetTextVal(F_ObjHandleT docId,
    F_TextRangeT *textRangep,
    IntT propNum,
    F_TypedValT *setVal);
```

Arguments

`docId`	The ID of the document containing the text range.
`textRangep`	The text range to set the property for. For information on specifying text locations, see "Getting and setting the insertion point or text selection" on page 321 of the *FDK Programmer's Guide*.
`propNum`	The number of the property to set.
`setVal`	The value to set the property to.

Returns

`VoidT`.

If `F_ApiSetTextVal()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadName`	Specified name is illegal
`FE_BadPropNum`	Specified property number is invalid
`FE_BadPropType`	Incorrect property type for this function
`FE_BadRange`	Specified text range is invalid
`FE_GenRuleAmbiguous`	General rule in structured document was ambiguous
`FE_GenRuleConnector Expected`	General rule in structured document missing connector
`FE_GenRuleItemExpected`	General rule in structured document missing rule item

FA_errno value	Meaning
FE_GenRuleLeftBracket Expected	General rule in structured document missing left bracket
FE_GenRuleMixed Connectors	General rule in structured document has mixed connectors
FE_GenRuleRightBracket Expected	General rule in structured document missing right bracket
FE_GenRuleSyntaxError	General rule in structured document has syntax error
FE_NotTextObject	Value must be the ID of an object that contains text, such as a paragraph (FO_Pgf) or a flow (FO_Flow)
FE_OffsetNotFound	Offset specified for the text location couldn't be found in the specified paragraph or text line
FE_OutOfRange	Specified property value is out of the legal range for the specified property
FE_ReadOnly	Property is read-only
FE_WrongProduct	Current FrameMaker product doesn't support the operation

Example

The following code turns change bars on for the selected text:

```
. . .
F_TextRangeT tr;
F_TypedValT setVal;
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId,FP_TextSelection);
if(!tr.beg.objId) return;

setVal.valType = FT_Integer;
setVal.u.ival = True;

/* Apply the text val to the text selection. */
F_ApiSetTextVal(docId, &tr, FP_ChangeBar, &setVal);
. . .
```

See also

"F_ApiGetTextPropVal()" on page 267, "F_ApiSetTextProps()" on page 445, and "F_ApiSetTextVal()" on page 452.

F_ApiSetUBytesByName()

`F_ApiSetUBytesByName()` sets an unsigned bytes (`F_UBytesT`) inset facet. The standard facets, EPSI and FrameImage, are examples of unsigned bytes facets.

To set an `F_UBytesT` facet, follow these steps:

1 *Call* `F_ApiSetUBytesByName()` *to set the facet data.*
If you are setting a facet with less than 10K of data, you need to call `F_ApiSetUBytesByName()` only once. If you are setting the facet with more than 10K of data, you should call `F_ApiSetUBytesByName()` multiple times, setting a chunk of the data each time. You can size the chunks between 0 and 10K. If you use larger chunks, the set operation goes faster. However, if the chunks are too large, you may overload your platform's interapplication communication mechanism.

2 *Call* `F_ApiSetUBytesByName()` *with* `propName` *set to an empty string (*`""`*).*
This lets the API know you have finished setting the facet.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSetUBytesByName(F_ObjHandleT docId,
       F_ObjHandleT objId,
       StringT propName,
       F_UBytesT *setVal);
```

Arguments

`docId`	The ID of the document containing the inset whose facet you want to set
`objId`	The ID of the inset whose facet you want to set
`propName`	The name of the facet to set
`setVal`	The data to which to set the facet

`F_UBytesT` is defined as:

```
typedef struct {
   UIntT len; /* The number of unsigned bytes */
   UByteT *val; /* The facet data */
} F_UBytesT;
```

Returns

`VoidT`.

If `F_ApiSetUBytesByName()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

See "Setting an F_UBytesT facet" on page 499 of the *FDK Programmer's Guide*.

See also

"F_ApiGetUBytesByName()" on page 273.

F_ApiShutDown()

`F_ApiShutDown()` closes an FDK client's connection with the API. Call it when the global FDK variable `FA_bailout` is set to a nonzero value. `F_ApiShutDown()` is useful only if you are providing a replacement for `F_ApiRun()`, for example, if you are writing a `main()` function for your client.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiShutDown(VoidT);
```

Arguments

None.

Returns

`VoidT`.

Example

The following code attempts to start the API. If it succeeds, it executes some code. If it fails, it exits and prints a message containing the error:

```
. . .
if (s = F_ApiStartUp(NULL))
    F_ApiErr(s);
else {
    /* . . . run some code . . . */
    }
F_ApiShutDown();
return(s!=NULL);
```

See also

"F_ApiStraddleCells()" on page 470.

F_ApiSilentPrintDoc()

`F_ApiSilentPrintDoc()` prints a document or a book using the default print settings. Default print settings are the settings that appear in the Print dialog box when the user attempts to print a document. `F_ApiSilentPrintDoc()` initializes the print page size and printer name if they don't have values.

To change a document or book's default print settings, set the document's print properties. For a list of print properties, see "Document print properties" on page 811 and "Book print properties" on page 736.

For example, to change the turn registration marks on when you print a document, use `F_ApiSetInt()` to set the document's `FP_RegistrationMarks` property. If you save the document or attempt to print it again within the same session, any changes you make to a document's print settings (except `FP_PrintStartPage` and `FP_PrintEndPage`) will appear the next time the user displays the Print dialog box for the document.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiSilentPrintDoc(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the book or document to print

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiSilentPrintDoc()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_SystemError`	Couldn't open or close the printer file
`FE_BadParameter`	Parameter has an invalid value

Example

The following function prints a range of pages. You set the range by specifying start and end values for the document; notice the different print properties used to set the page range for different numbering styles.

```
VoidT doSilentPrint(F_ObjHandleT docId, IntT start, IntT end)
{
StringT pageString;
F_ObjHandleT pageId;
IntT i = 0;

/* First specify that you will print a range of pages */
F_ApiSetInt(FV_SessionId, docId, FP_PrintScope, FV_PR_RANGE);

/* If page numbering is numeric, use values of start and end. */
if(F_ApiGetInt(FV_SessionId,docId,FP_PageNumStyle)==
                          FV_PAGE_NUM_NUMERIC) {
  F_ApiSetInt(FV_SessionId, docId, FP_PrintStartPage, start);
  F_ApiSetInt(FV_SessionId, docId, FP_PrintEndPage, end);
} else {
/* For non-numeric page nums, convert start and end to */
/* the corresponding page name strings. */
  pageId = F_ApiGetId(
                FV_SessionId, docId, FP_FirstBodyPageInDoc);
  while(pageId) {
    i++;
    if(i == start) {
       pageString=F_ApiGetString(docId,pageId,FP_PageNumString);
       F_ApiSetString(FV_SessionId, docId,
                        FP_PrintStartPageName, pageString);
       if(!F_StrIsEmpty(pageString))
          F_ApiDeallocateString(&pageString);
    } else if(i == end) {
       pageString=F_ApiGetString(docId,pageId,FP_PageNumString);
       F_ApiSetString(FV_SessionId, docId,
                        FP_PrintEndPageName, pageString);
       if(!F_StrIsEmpty(pageString))
```

```
            F_ApiDeallocateString(&pageString);
    }
    pageId = F_ApiGetId(docId, pageId, FP_PageNext);
  }
}
F_ApiSilentPrintDoc(docId);
}
```

See also

"F_ApiOpen()" on page 355.

F_ApiSimpleGenerate()

`F_ApiSimpleGenerate()` generates files for a book. It performs the same operation as choosing Update Book from the book Edit menu. The book and its generated files must be set up before you call `F_ApiSimpleGenerate()`.

You may want to call `F_ApiUpdateXRefs()` to update all cross references before calling `F_ApiSimpleGenerate()`.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiSimpleGenerate(F_ObjHandleT bookId,
    IntT interactive,
    IntT makeVisible);
```

Arguments

`bookId`	The ID of the book for which to generate files.
`interactive`	Specifies whether to display warnings, messages, and the book error log to the user. `True` displays messages and warnings.
`makeVisible`	Specifies whether to display the generated files (`True` displays the files).

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiSimpleGenerate()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Bad book ID
`FE_BadOperation`	The book is not self-consistent (book generates data in one file that is source data for another generated file, or page count continually changes for this operation); there is a duplicate file in the book; all files in the book are generated files
`FE_BadParameter`	`bookId` was not a valid book ID
`FE_SystemError`	Couldn't allocate memory, or couldn't open or save one of the files in the book
`FE_WrongProduct`	The current product is not supported

Example

The following code generates files for a book:

```
. . .
F_ObjHandleT bookId;
. . .
F_ApiSimpleGenerate(bookId, False, False);
. . .
```

See also

"F_ApiUpdateBook()" on page 480 and "F_ApiUpdateXRefs()" on page 492.

Structured F_ApiSimpleImportElementDefs()

`F_ApiSimpleImportElementDefs()` imports element definitions and the format change list catalog from an EDD or Structured FrameMaker document or book to a Structured FrameMaker document or book.

If you import element definitions to a book, `F_ApiSimpleImportElementDefs()` imports element definitions to each book component for which the `FP_ImportFmtInclude` property is set to `True`.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiSimpleImportElementDefs(F_ObjHandleT docOrBookId,
    F_ObjHandleT fromDocOrBookId,
    IntT importFlags);
```

Arguments

`docOrBookId`	The ID of the book or document to which to import element definitions.
`fromDocOrBookId`	The ID of the document or book from which to import element definitions.
`importFlags`	Specify how to import formats. See the following table for the flags that you can OR into this parameter.

The following table lists flags that you can OR into the `importFlags` parameter:

Flag	Meaning
`FF_IED_REMOVE_OVERRIDES`	Clear format overrides.
`FF_IED_REMOVE_BOOK_INFO`	If `docOrBookId` specifies a document, clear formatting inherited from a parent book.
`FF_IED_DO_NOT_IMPORT_EDD`	If the document specified by `fromDocOrBookId` is an EDD, don't treat it as an EDD; just import its element catalog.
`FF_IED_NO_NOTIFY`	Do not issue the `FA_Note_PreImportElemDefs` or `FA_Note_PostImportElemDefs` notifications.

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiSimpleImportElementDefs()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product isn't supported
`FE_BadDocId`	Invalid book or document ID

Example

The following code imports the Element Catalog from one document to another and reformats the document using the new catalog:

```
. . .
F_ObjHandleT fromDocId, toDocId;
. . .
F_ApiSimpleImportElementDefs(toDocId, fromDocId,
          FF_IED_REMOVE_OVERRIDES | FF_IED_DO_NOT_IMPORT_EDD);
. . .
```

F_ApiSimpleImportFormats()

`F_ApiSimpleImportFormats()` imports formats from a document to a document or a book. If you import formats to a book, `F_ApiSimpleImportFormats()` imports formats to each book component for which the `FP_ImportFmtInclude` property is set to `True`.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiSimpleImportFormats(F_ObjHandleT bookId,
   F_ObjHandleT fromDocId,
   IntT formatFlags);
```

Arguments

`bookId`	The ID of the book or document to which the formats are to be imported.
`fromDocId`	The ID of the document from which to import formats.
`formatFlags`	Bit field specifying which formats to import. Specify 0 for the default flags.

You can OR the following values into the `formatFlags` parameter to specify which formats to import.

This value	To
`FF_UFF_COLOR`	Import colors
`FF_UFF_COMBINED_FONTS`	Import combined fonts
`FF_UFF_COND`	Import conditions
`FF_UFF_DOCUMENT_PROPS`	Import document properties

This value	To
FF_UFF_FONT	Import Character Catalog formats
FF_UFF_MATH	Import equation settings
FF_UFF_PAGE	Import page layouts
FF_UFF_PGF	Import Paragraph Catalog formats
FF_UFF_REFPAGE	Import reference pages
FF_UFF_REMOVE_EXCEPTIONS	Remove exception formats from target documents
FF_UFF_REMOVE_PAGE_BREAKS	Remove all forced page breaks from target documents
FF_UFF_TABLE	Import Table Catalog formats
FF_UFF_VAR	Import variable formats
FF_UFF_XREF	Import cross-reference formats

Returns

FE_Success if it succeeds, or an error code if an error occurs.

If F_ApiSimpleImportFormats() fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
FE_WrongProduct	Current FrameMaker product doesn't support books
FE_BadDocId	Invalid book or document ID
FE_Canceled	User canceled the operation
FE_FailedState	The FrameMaker product failed to open one or more of the book's document files during the import operation

Example

The following code imports cross-reference and Paragraph Catalog formats from a document to all the documents in a book:

```
. . .
F_ObjHandleT docId, bookId;
. . .
F_ApiSimpleImportFormats(bookId, docId,
                            FF_UFF_XREF | FF_UFF_PGF);
. . .
```

F_ApiSimpleNewDoc()

`F_ApiSimpleNewDoc()` creates a new document from a specified template.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiSimpleNewDoc(StringT templateName,
    IntT interactive);
```

Arguments

`templateName`	The absolute pathname of the template to use. For information on how to specify a pathname on a platform, see the *FDK Platform Guide* for that platform.
`interactive`	Specifies whether the FrameMaker product displays messages and warnings to the user.

Returns

The ID of the new document if it is successful, or `0` if it fails.

IMPORTANT: *If you call* `F_ApiSimpleNewDoc()` *with* `interactive` *set to* `True` *and the user clicks Portrait, Custom, or Landscape in the New dialog box,* `F_ApiSimpleNewDoc()` *does not create a document. It returns* `0` *and sets* `FA_errno` *to* `FE_WantsPortrait`, `FE_WantsCustom`, *or* `FE_WantsLandscape`. *It is up to your client to create a portrait, custom, or landscape document. For information on creating custom documents, see* *"F_ApiCustomDoc()" on page 119.*

If `F_ApiSimpleNewDoc()` fails, the API assigns an error code to `FA_errno`. The error codes for `F_ApiSimpleNewDoc()` are the same as those for `F_ApiOpen()`. For the list of these error codes, see "F_ApiOpen()" on page 355.

Example

The following code creates a document from a template named `Report1` in the `/fmtemplates/Reports` directory. It instructs the FrameMaker product to prompt the user with messages or warnings that occur.

```
. . .
F_ObjHandleT docId;

docId = F_ApiSimpleNewDoc("/fmtemplates/Reports/Report1", True);
. . .
```

See also

"F_ApiOpen()" on page 355.

F_ApiSimpleOpen()

`F_ApiSimpleOpen()` opens a document or book.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiSimpleOpen(StringT fileName,
   IntT interactive);
```

Arguments

`fileName`	The absolute pathname of the file to open. For information on specifying a pathname for a platform, see the *FDK Platform Guide* for that platform.
`interactive`	Specifies whether the FrameMaker product displays messages and warnings to the user. `True` instructs the FrameMaker product to display messages and warnings.

If you call `F_ApiSimpleOpen()` with `interactive` set to `True`, the FrameMaker product displays the Open dialog box. It uses the path specified by the session property, `FP_OpenDir`, as the default path. If a warning or error condition arises, the FrameMaker product notifies the user. For example, if a document uses fonts that are not available, the FrameMaker product displays a dialog box that allows the user to cancel the operation or to continue and remap the fonts.

If you set `interactive` to `False`, the FrameMaker product does not display the Open dialog box or other messages and warnings. If it is necessary to modify a file to continue opening it, `F_ApiSimpleOpen()` aborts the operation without notifying the user, and returns `0`.

You can't use `F_ApiSimpleOpen()` to open filterable files. To open filterable files, use `F_ApiOpen()`.

Returns

The ID of the opened document, or `0` if an error occurs.

If `F_ApiSimpleOpen()` fails, the API assigns an error code to `FA_errno`. The error codes for `F_ApiSimpleOpen()` are the same as those for `F_ApiOpen()`. For the list of these error codes, see "F_ApiOpen()" on page 355.

Example

The following code opens a document named `/tmp/my.doc`. It displays the document's ID in a dialog box. It does not prompt the user with other messages or warnings.

```
. . .
#include "futils.h"
F_ObjHandleT docId;
UCharT msg[256];

docId = F_ApiSimpleOpen("/tmp/my.doc", False);

if (!docId) F_ApiAlert("Can't open my.doc.",
        FF_ALERT_CONTINUE_NOTE);
else{
    F_Sprintf(msg, "The opened document's ID is 0x%x.", docId);
    F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
    }
. . .
```

See also

"F_ApiOpen()" on page 355.

F_ApiSimpleSave()

`F_ApiSimpleSave()` saves a document or book.

Synopsis

```
#include "fapi.h"
. . .
F_ObjHandleT F_ApiSimpleSave(F_ObjHandleT docId,
    StringT saveAsName,
    IntT interactive);
```

Arguments

`docId`	The ID of the document or book to save.
`saveAsName`	The absolute pathname to save the document or book to. For information on how to specify a pathname for a particular platform, see the *FDK Platform Guide* for that platform.
`interactive`	Specifies whether the FrameMaker product displays messages and warnings to the user. `True` displays messages and warnings.

If you set `interactive` to `False` and you specify the document or book's current name for `saveAsName`, the FrameMaker product saves the document or book under its current name. If you specify another filename for `saveAsName`, the FrameMaker product saves the document or book to that filename. If you specify an empty string (`""`), the FrameMaker product doesn't save the file. Instead it sets `FA_errno` to `FE_BadParameter`.

If you set `interactive` to `True`, the FrameMaker product displays the Save dialog box and allows the user to choose a filename. The document or book's current name appears as the default name.

Returns

The ID of the document it saved, or `0` if it is not successful.

If `F_ApiSimpleSave()` fails, the API assigns an error code to `FA_errno`. The error codes for `F_ApiSimpleSave()` are the same as those for `F_ApiSave()`. For the list of these error codes, see "F_ApiSave()" on page 399.

Example

The following code opens a document named `my.doc` from the `/tmp` directory and saves it as `mynew.doc`. It does not use the interactive mode, so the FrameMaker product doesn't prompt the user if a file named `mynew.doc` already exists, or if anything goes wrong.

```
. . .
F_ObjHandleT docId;

docId = F_ApiSimpleOpen("/tmp/my.doc", False);
if (docId) F_ApiSimpleSave(docId, "/tmp/mynew.doc", False);
. . .
```

See also

"F_ApiOpen()" on page 355 and "F_ApiSave()" on page 399.

F_ApiSleep()

`F_ApiSleep()` suspends client and FrameMaker product operation for a specified number of seconds.

IMPORTANT: *Do not call* `sleep()` *in your client. Use* `F_ApiSleep()` *instead.*

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiSleep(IntT seconds);
```

Arguments

`seconds`	The number of seconds to suspend FrameMaker product and client operation

Returns

The number of seconds remaining in the sleep period.

If `F_ApiSleep()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_Transport`	A transport error occurred

Example

The following code suspends client and FrameMaker product operation for two seconds:

```
. . .
F_ApiSleep(2);
. . .
```

See also

"F_ApiUSleep()" on page 494.

Structured F_ApiSplitElement()

`F_ApiSplitElement()` splits the structural element containing the insertion point into two elements at the insertion point. The insertion point must be inside the element you want to split.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiSplitElement(F_ObjHandleT docId);
```

Arguments

`docId` The document containing the selected elements

Returns

`VoidT`.

If `F_ApiSplitElement()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Invalid document ID
`FE_BadSelectionForOperation`	Current text selection is invalid for this operation
`FE_WrongProduct`	Current FrameMaker product isn't supported

Example

The following code sets the insertion point and splits the element that contains the insertion point:

```
. . .
F_TextRangeT tr;
F_ObjHandleT docId;
. . .
F_ApiSetTextRange(FV_SessionId, docId, FP_TextSelection, &tr);
F_ApiSplitElement(docId);
. . .
```

F_ApiStraddleCells()

`F_ApiStraddleCells()` straddles the specified cells in a table. The cells you straddle must all be from the same type of row. You can't straddle a set of cells that are in both heading and body rows or footing and body rows. Also, the cells you straddle must be unstraddled; you cannot use this function to further straddle cells that are already straddled.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiStraddleCells(F_ObjHandleT docId,
    F_ObjHandleT cellId,
    IntT heightInRows,
    IntT widthInCols);
```

Arguments

`docId`	The ID of the document containing the table
`cellId`	The ID of the first (leftmost and uppermost) cell in the range of cells to straddle
`heightInRows`	The number of cells to straddle vertically
`widthInCols`	The number of cells to straddle horizontally

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiStraddleCells()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product does not support tables
`FE_BadOperation`	Parameters specify an action that is invalid
`FE_BadDocId`	Invalid document ID
`FE_BadParameter`	Parameter has an invalid value
`FE_BadObjId`	Invalid cell ID

Example

The following code straddles the first two cells in the first row of the first table in the active document:

```
. . .
F_ObjHandleT docId, tableId, firstrowId, cellId;

/* Get IDs of document, table, and first cell. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tableId = F_ApiGetId(FV_SessionId, docId, FP_FirstTblInDoc);
firstrowId = F_ApiGetId(docId, tableId, FP_FirstRowInTbl);
cellId = F_ApiGetId(docId, firstrowId, FP_FirstCellInRow);

if (F_ApiGetInt(docId, tableId, FP_TblNumCols) < 2)
   F_ApiAlert("Not enough columns!", FF_ALERT_CONTINUE_NOTE);
else F_ApiStraddleCells(docId, cellId, 1, 2);
. . .
```

See Also

".F_ApiUnStraddleCells()" on page 477.

F_ApiStringLen()

`F_ApiStringLen()` returns the length of a string.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiStringLen(ConStringT s);
```

Arguments

`s` The string for which to return the length

Returns

The length of the specified string.

Example

The following code prints the length of the string `supercalifragilisticexpialidocious`:

```
. . .
F_Printf(NULL, "The string length is %d\n",
    F_ApiStringLen("supercalifragilisticexpialidocious"));
. . .
```

Structured F_ApiTextLocToElementLoc()

F_ApiTextLocToElementLoc() returns the element location structure that corresponds to the current text location.

Synopsis

```
#include "fapi.h"
. . .
F_ElementLocT F_ApiTextLocToElementLoc (F_ObjHandleT docId,
                const F_TextLocT *tlocp);
```

Arguments

docId	The ID of the document containing the element
tlocp	The text location structure to convert

Returns

An F_ElementLocT structure specifying an element location. The F_ElementLocT structure is defined as:

```
typedef struct {
   F_ObjHandleT parentId; /* Parent element ID. */
   F_ObjHandleT childId; /* Child element ID. */
   IntT offset; /* Offset within child/parent element. */
} F_ElementLocT;
```

If F_ApiTextLocToElementLoc() fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
FE_BadDocId	Invalid document ID
FE_BadParameter	tlocp was empty or a parameter was improperly specified
FE_WrongProduct	Current FrameMaker product doesn't support the operation

Example

The following code converts a text location to an element location in order to determine which elements can legally be inserted at the location:

```
. . .
F_ObjHandleT docId, elemId;
F_TextRangeT tRange;
F_TextLocT tloc;
F_ElementLocT elemLoc;
StringT elemName;
. . .
tRange = F_ApiGetTextRange(FV_SessionId, docId,
             FP_TextSelection);
tloc = tRange.beg
elemLoc = F_ApiTextLocToElementLoc(docId, &tloc);
/* get Id of the parent element's element definition */
elemId = F_ApiGetId(docId, elemLoc.parentId, FP_ElementDef);
elemName = F_ApiGetString(docId, elemId, FP_Name);
/* process to determine which elements can be inserted */
. . .
```

See also

"F_ApiTextLocToElementLoc()" on page 473.

See also

"F_ApiElementLocToTextLoc()" on page 160.

F_ApiUndoCancel()

`F_ApiUndoCancel()` clears both the undo and redo stacks in the document specified by docId. If an undo checkpoint has been started and not ended in this document, this call cancels the grouping operation.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiUndoCancel(F_ObjHandleT docId);
```

Arguments

`docId`	The ID of the document containing the undo and the redo stacks

Returns

On success, sets `FA_errno` to `FE_Success`. Otherwise, sets `FA_errno` to `FE_BadDocId` (invalid document ID).

Example

This example demonstrates how to clear the undo stack when undoing an action could corrupt an external file. In this case, a string is deleted from an external database permanently when selected text is deleted in FrameMaker. If you cannot restore the deleted string in the database, undoing the delete action from a FrameMaker document would result in database corruption. To protect against this, the example uses `F_ApiUndoCancel` to clear all user actions saved in the undo stack.

```
F_TextRangeT tr;
F_TextItemsT textItems;
StringT string = NULL;

tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
textItems = F_ApiGetTextForRange(docId, &tr, FTI_String);
string = CreateStringFromTextItems(textItems);

F_ApiDeleteText(docId, &tr);
DeleteTextFromDatabasePermanently(string);
F_ApiUndoCancel(docId);
```

F_ApiUndoEndCheckPoint()

`F_ApiUndoEndCheckPoint()` marks the ending point of a series of API calls that are to be treated as a single undoable operation. The docid must specify the same document as the corresponding call to `F_ApiUndoStartCheckpoint`.

If any API call in the series clears the undo stack, the stack is cleared after the end checkpoint is reached.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiUndoEndCheckPoint(F_ObjHandleT docId);
```

Arguments

`docId`	The ID of the document in which the ending point i to be marked.

Returns

On success, sets `FA_errno` to `FE_Success`. Otherwise, sets `FA_errno` to one of the following:

- `FE_BadDocId` (invalid document ID).
- `FE_FDKUndoNotAllowed`: whenever fdk undo recording flag is off and checkpoint for undo is requested

Example

This example combines two API calls (`F_ApiSetTextProps` and `F_ApiAddText`) into one undoable action. The specified command name, "Add Big Red Text"appears in the Undo menu and the Command History palette, rather than the two command names "Set Text Property" and "Add Text".

```
#define pts (MetricT)65536
F_TextRangeT tr;
F_ObjHandleT colorId;
F_PropValsT props;
IntT i;

tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
colorId = F_ApiGetNamedObject(docId, FO_Color, (StringT)"Red");
props = F_ApiGetTextProps(docId, &tr.beg);
i = F_ApiGetPropIndex(&props, FP_Color);
props.val[i].propVal.u.ival = colorId;
i = F_ApiGetPropIndex(&props, FP_FontSize);
props.val[i].propVal.u.ival = 100 * pts;


F_ApiUndoStartCheckpoint(docId, "Add Big Red Text");
F_ApiSetTextProps(docId, &tr, &props);
F_ApiDeallocatePropVals(&props);
F_ApiAddText(docId, &tr.beg, (StringT)"Big Red Text!");
F_ApiUndoEndCheckpoint(docId);
```

F_ApiUndoStartCheckPoint()

`F_ApiUndoStartCheckPoint()` records the starting point of a series of API calls that are to be treated as a single undoable operation in the document specified by docId.

If there is no corresponding call to F_APIUndoEndCheckpoint, all subsequent API calls are grouped into a single undoable operation.

You cannot nest checkpoints. A second call to this function that appears before the corresponding call to F_APIUndoEndCheckpoint is ignored.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiUndoStartCheckPoint(F_ObjHandleT docId, ConStringT
description);
```

Arguments

docId	The ID of the document in which the ending point i to be marked
Description	The string that appears in the Undo and Redo menus and the Command History palette

Returns

On success, sets FA_errno to FE_Success. Otherwise, sets FA_errno to one of the following:

- FE_BadDocId (invalid document ID).
- FE_FDKUndoNotAllowed: whenever fdk undo recording flag is off and checkpoint for undo is requested

Example

See the example provided in "F_ApiUndoEndCheckPoint()" on page 475

.F_ApiUnStraddleCells()

F_ApiUnStraddleCells() unstraddles the specified cells in a table.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiUnStraddleCells(F_ObjHandleT docId,
   F_ObjHandleT cellId,
   IntT heightInRows,
   IntT widthInCols);
```

Arguments

`docId`	The ID of the document containing the table
`cellId`	The ID of the first (leftmost and uppermost) cell in the range of cells to unstraddle
`heightInRows`	The number of cells to unstraddle vertically
`widthInCols`	The number of cells to unstraddle horizontally

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiUnStraddleCells()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product does not support tables
`FE_BadOperation`	Parameters specify an action that is invalid
`FE_BadDocId`	Invalid document ID
`FE_BadParameter`	Parameter has an invalid value
`FE_BadObjId`	Invalid cell ID

Example

The following code unstraddles the first two cells in the first row of the first table in the active document:

```
. . .
F_ObjHandleT docId, tableId, firstrowId, cellId;

/* Get IDs of document, table, and first cell. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tableId = F_ApiGetId(FV_SessionId, docId, FP_FirstTblInDoc);
firstrowId = F_ApiGetId(docId, tableId, FP_FirstRowInTbl);
cellId = F_ApiGetId(docId, firstrowId, FP_FirstCellInRow);

F_ApiUnStraddleCells(docId, cellId, 1, 2);
. . .
```

See also

"F_ApiStraddleCells()" on page 470.

Structured F_ApiUnWrapElement()

`F_ApiUnWrapElement()` removes the selected structural elements, but leaves their contents and child elements intact in the document. `F_ApiUnWrapElement()` does not remove all the elements in the selection, just the top-level elements.

IMPORTANT: *At least one structural element must be selected when* `F_ApiUnWrapElement()` *is called.* `F_ApiUnWrapElement()` *has no effect on object elements.*

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiUnWrapElement(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document containing the selected elements

Returns

`VoidT`.

If `F_ApiUnWrapElement()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation
`FE_BadDocId`	Invalid document ID
`FE_BadSelectionForOperation`	Current text selection is invalid for this operation

Example

The following code sets the text selection and unwraps the elements in it:

```
. . .
F_TextRangeT tr;
F_ObjHandleT docId;
. . .
F_ApiSetTextRange(FV_SessionId, docId, FP_TextSelection, &tr);
F_ApiUnWrapElement(docId);
. . .
```

See also

"F_ApiWrapElement()" on page 496.

F_ApiUpdateBook()

F_ApiUpdateBook() performs Update Book commands.

`F_ApiUpdateBook()` allows you to specify a script (property list) telling the FrameMaker product how to update the book and how to deal with error and warning conditions. For example, you can specify whether to abort or to continue updating a book if it contains view-only documents.

Synopsis

```
#include "fapi.h"
. . .
ErrorT F_ApiUpdateBook(F_ObjHandleT bookId,
   F_PropValsT *updateParamsp,
   F_PropValsT **updateReturnParamspp);
```

Arguments

`bookId`	The ID of the book to update
`updateParamsp`	A property list telling the FrameMaker product how to update the book and how to respond to errors and other conditions. To use the default list, specify `NULL`.
`updateReturnParamspp`	A property list that provides information about how the FrameMaker product updated the book. It must be initialized before you call `F_ApiUpdateBook()`.

IMPORTANT: *Always initialize the pointer to the property list that you specify for* `updateReturnParamspp` *to* `NULL` *before you call* `F_ApiUpdateBook()`.

To get a property list to specify for the `updateParamsp` parameter, use `F_ApiGetUpdateBookDefaultParams()` or create the list from scratch. For a list of all the properties an Update Book script can include, see "F_ApiGetUpdateBookDefaultParams()" on page 278. For an example of how to create a property list from scratch, see "Example" on page 243 of the *FDK Programmer's Guide*.

While updating a book, you can post messages to the book error log. To do so, you use `F_ApiCallClient()` to pass your messages to the BookErrorLog client.

Returns

An `ErrorT`, which has the same value as `FA_errno`.

The property list that `updateReturnParamspp` is set to has the properties shown in the following table.

Property	Meaning and possible values
`FS_UpdateBookStatus`	A bit field to indicate what happened during the update. See the table below for a list of possible flags.

If an error occurs, the FA_errno global variable indicates the result of a call to F_ApiUpdateBook(). The following table lists the possible status flags and the FA_errno value associated with them.

FA_errno values	Possible FS_UpdateBookStatus flags
FE_BadDocId	None
FE_BadOperation	FV_BookNotSelfConsistent: The book is not self-consistent (book generates data in one file that is source data for another generated file, or page count continually changes for this operation)
	FV_DuplicateFileInBook: one or more files in the book is a duplicate of another file
	FV_NoNonGeneratedFilesInBook: the only files in the book are generated files
FE_BadParameter	FV_BadUpdateBookFileId: specified book ID is invalid
	FV_BadUpdateBookScriptValue: the update book script contained an invalid property value
FE_Canceled FE_CanceledByClient	FV_CancelInconsistentNumPropsInFileInBook: one or more of the book's document files has numbering properties that are inconsistent with the properties stored in the book
	FV_CancelNonFMFileInBook: one or more of the book's document files is not a FrameMaker product file
	FV_CancelViewOnlyFileInBook: one or more of the book's document files is view-only
	FV_UserCanceledUpdateBook: the user cancelled the update operation
FE_SystemError	FV_FileInBookNotOpened: one or more files in the book could not be opened
	FV_FileInBookNotSaved: one or more files in the book could not be saved
	FV_TooManyWindowsUpdateBook: too many windows were open for the currently available memory

To determine if a particular FS_UpdateBookStatus bit is set, use F_ApiCheckStatus(). For more information, see "F_ApiCheckStatus()" on page 96.

After you are done with the property lists you use to call F_ApiUpdateBook(), use F_ApiDeallocatePropVals() to deallocate the memory they use.

Example

The following code updates the active book. It creates a property list that instructs the FrameMaker product to update file numbering and cross-references in the book.

```
. . .
#include "futils.h"

F_PropvalsT params, *returnp = NULL;
F_ObjHandldleT bookId;
ErrorT err;

#define ALERT_USR          0
#define NUM_PROP_VALS      1

/* Allocate memory for the Update Book script. */
params = F_ApiAllocatePropVals(NUM_PROP_VALS);
if(params.len == 0)
   return;

/* Alert user for various conditions */
params.val.[ALERT_USR].propIdent.num = FS_AlertUserAboutFailure;
params.val.[ALERT_USR].propIdent.valType = FT_Integer;
params.val.[ALERT_USR].propIdent.u.ival = True;

/* Get book ID and update the book. */
bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook);
if(!bookId)
   return;
err = F_ApiUpdateBook(bookId, &params, &returnp);

/* Free memory for the property values. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnp);

. . .
```

See also

"F_ApiGetUpdateBookDefaultParams()" on page 278 and "F_ApiCheckStatus()" on page 96.

F_ApiUpdateDITAReference()

Updates a DITA object.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiUpdateDITAReference(F_ObjHandleT docId, F_ObjHandleT
elemId, IntT objType);
```

Arguments

`docId`	The ID of the document containing the object.
`elemId`	The ID of the element representing the DITA object to be updated.
`objType`	Can have one of the following values: `FV_DITAObjTypeAuto`: Automatically determines the DITA object type of the element and updates accordingly. `FV_DITAObjTypeConref`: The element to be updated is a DITA conref. If not, does not update and returns an error. `FV_DITAObjTypeLink`: The element to be updated is a DITA link. If not, does not update and returns an error. `FV_DITAObjTypeTopicref`: The element to be updated is a DITA topicref. If not, does not update and returns an error. `FV_DITAObjTypeTopicsetref`: The element to be updated is a DITA topicsetref. If not, does not update and returns an error.

Returns

If `F_ApiUpdateDITAReference()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation.
`FE_BadDocId`	The Document ID provided is invalid.
`FE_BadElementId`	The Element ID provided is invalid.
`FE_NonDITADocument`	The Document provided is not a DITA document.
`FE_BadParameter`	The `objType` provided is invalid or the `objType` is not valid for the type of DITA document provided.
`FE_UpdateDITAReferenceFailedInvalidElementType`	Update operation failed because either the element specified is not a reference type of element or it does not match the specified object type.

FA_errno value	Meaning
FE_UpdateDITAReferenceFailedCannotResolveReference	Update operation failed because the reference cannot be resolved.
FE_UpdateDITAReferenceFailedCannotFindReferencedFile	Update operation failed because the referenced file cannot be found at the specified location.
FE_UpdateDITAReferenceFailedCannotOpenReferencedFile	Update operation failed because the referenced file cannot be opened from the specified location.
FE_UpdateDITAReferenceFailedCannotConvertToFMObject	Update operation failed because the corresponding FM object cannot be created.
FE_UpdateDITAReferenceFailed	Update operation failed.

F_ApiUpdateDITAReferences()

Updates all DITA references of the specified type.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiUpdateDITAReferences(F_ObjHandleT docId, IntT flag);
```

Arguments

`docId`	The ID of the document containing the inset.
`flag`	The available flags and their values are as follows: `FF_DITAUpdateAllConrefs`: 0x01 `FF_DITAUpdateAllXrefs`: 0x02 `FF_DITAUpdateAllLinks`: 0x04 `FF_DITAUpdateAllTopicrefs`: 0x08 `FF_DITAUpdateAllTopicsetrefs`: 0x10 `FF_DITAUpdateAllReferences`: `FF_DITAUpdateAllConrefs | FF_DITAUpdateAllXrefs | FF_DITAUpdateAllLinks | FF_DITAUpdateAllTopicrefs | FF_DITAUpdateAllTopicsetrefs`

In case of multiple reference types being updated, the objects will be updated in the following sequence:

API Sequence	Client Sequence
`Topicrefs`	`Topicrefs`
`Xrefs`	`Topicsetrefs`
`Links`	`Conrefs`
`Topicsetrefs`	`Xrefs`
`Conrefs`	`Links`

Corresponding to this API, the notification `FA_Note_UpdateDITAReferences` is used separately once for each reference type that is to be updated. These notifications can also be handled in the user client.

Returns

If `F_ApiUpdateDITAReferences()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support the operation.
`FE_BadDocId`	The document ID provided is invalid.
`FE_NonDITADocument`	The document provided is not a DITA document.

F_ApiUpdateKeyDefinition()

Updates the specified key definition field for the specified key in the specified key catalog.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiUpdateKeyDefinition(F_ObjHandleT keyCatalogId, StringT
key, IntT keyField, const F_TypedValT *valuep)
```

Arguments

`keyCatalogId`	The ID of the key catalog for which to update the key definiton.
`key`	The tag of the key for which the key definition is being updated.
`keyField`	The key field (or key information) that is being updated.
`valuep`	The value to update the `keyField` to.

The valid `keyField` values and the corresponding value type are as follows:

keyField	Value type
`FV_KeydefKeyTarget`	`FT_String`
`FV_KeydefKeySrcFile`	`FT_String`
`FV_KeydefKeySrcType`	`FT_Integer`
`FV_KeydefKeyVarList`	`FT_Vals`

keyField	Value type
FV_KeydefKeyDefaultText	FT_String
FV_KeydefKeyFoundInRefFile	FT_Integer
FV_KeydefKeyInValid	FT_Integer
FV_KeydefKeyAttrs	FT_AttributesEx

Returns

If F_ApiUpdateTextInset() fails, the API assigns one of the following values to FA_errno.

FA_errno value	Meaning
FE_BadObjId	The ID provided does not specify a key catalog.
FE_BadKey	The key provided is not valid.
FE_KeyDefinitionDoesNotExist	The definition for the specified key is not available in the key catalog.
FE_BadValue	The value is either not specified or it is not as expected for the specified keyField.
FE_ReadOnly	(only for keyField=FV_KeydefKeyTag or FV_KeydefKeyDuplicate). The key field cannot be changed or updated.
FE_InvAttribute	(only for keyField=FV_KeydefKeyAttrs) The Attribute information provided is not valid.
FE_WrongProduct	(only for keyField=FV_KeydefKeyAttrs) Current FrameMaker product doesn't support the operation.
FE_BadKeyField	The key field provided is not valid.

F_ApiUpdateTextInset()

F_ApiUpdateTextInset() updates the contents of a stale text inset. It determines whether an inset is stale by comparing the inset's FP_TiLastUpdate property with the modification date of the inset's source file.

F_ApiUpdateTextInset() does not update a text inset unless it is stale. To make a text inset stale, set its FP_TiLastUpdate property to 0.

F_ApiUpdateTextInset() does not update graphic insets (FO_Inset objects).

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiUpdateTextInset(F_ObjHandleT docId,
   F_ObjHandleT textInsetId);
```

Arguments

`docId`	The ID of the document containing the inset.
`textInsetId`	The ID of the text inset to update. To update all stale insets in the document that are marked for automatic update, specify `0`.

Returns

`FE_Success` if it succeeds or `FE_SomeUnresolved` if some text insets were unresolved.

If `F_ApiUpdateTextInset()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Bad document or book ID
`FE_BadFileType`	The inset specifies a file that does not match the import type (for example, the inset imports a binary document but the file is a text file or doesn't exist)
`FE_SomeUnresolved`	Some text insets were unresolved
`FE_WrongProduct`	Product doesn't support the specified operation
`FE_SystemError`	Couldn't allocate memory

F_ApiUpdateVariables()

`F_ApiUpdateVariables()` updates all the variables in a document. It performs the same operation as clicking Update in the Variable dialog box.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiUpdateVariables(F_ObjHandleT docId);
```

Arguments

`docId` The ID of the document in which to update variables

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiUpdateVariables()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_BadDocId`	Bad document or book ID
`FE_WrongProduct`	Product doesn't support the specified operation
`FE_SystemError`	Couldn't allocate memory

Example

The following code updates the variables in the active document:

```
. . .
F_ApiUpdateVariables(F_ApiGetId(0, FV_SessionId, FP_ActiveDoc));
. . .
```

F_ApiUpdateXRef()

Updates a specified cross-reference in the document.

Synopsis

```
#include "fapi.h"
...
IntT F_ApiUpdateXRef(F_ObjHandleT docId,
F_ObjHandleT srcDocId,
F_ObjHandleT xrefId);
```

Arguments

`docId`	The ID of the document that contains the cross-reference.
`srcDocId`	The ID of the source document that the cross-reference references.
`xrefId`	The ID of the cross-reference to be updated.

Returns

FE_Success if it succeeds, or an error code if an error occurs.

If `F_ApiUpdateXRef()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno Value	Meaning
`FE_BadDocId`	Invalid document ID.
`FE_BadXRefSrcDoc`	Invalid source document ID.
`FE_BadObjId`	Invalid cross-reference ID.
`FE_XRefUnresolved`	FrameMaker cannot update the cross-reference, as it cannot find the source.

Example

The following code updates the cross-reference to the same or another document.

```
. . .
F_ObjHandleT xrefId = F_ApiGetId(FV_SessionId, docId,
                            FP_FirstXRefInDoc);
F_ObjHandleT srcDocId;
StringT srcFile;

/* Open the source document */
srcFile = F_ApiGetString(docId, xrefId, FP_XRefFile);

if(F_StrIsEmpty(srcFile))
   srcDocId = docId;
else
{
   F_PropValsT params, *returnp = NULL;

   /* Set the open & return parameters */
   . . .
   srcDocId = F_ApiOpen(srcFile, &params, &returnp);
}

F_ApiUpdateXRef(docId, srcDocId, xrefId);
. . .
```

F_ApiUpdateXRefs()

`F_ApiUpdateXRefs()` updates the cross-references in a document. It performs the same operation as clicking Update in the Cross-Reference window.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiUpdateXRefs(F_ObjHandleT docId,
   IntT updateXRefFlags);
```

Arguments

`docId`	The ID of the document in which to update cross-references.
`updateXRefFlags`	Flags to indicate which cross-references to update. See the following table for values.

You can OR the values listed in the following tables into the `updateXRefFlags` argument.

This value	For
`FF_XRUI_FORCE_UPDATE`	Updates all cross-references, regardless of whether the source document has changed
`FF_XRUI_INTERNAL`	Only internal cross-references
`FF_XRUI_OPEN_DOCS`	Only cross-references whose sources are in open documents
`FF_XRUI_CLOSED_DOCS`	Only cross-references whose sources are in closed documents
`FF_XRUI_EVERYTHING`	All cross-references

Returns

`FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiUpdateXRefs()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product doesn't support the requested operation
`FE_BadDocId`	Invalid document ID
`FE_SomeUnresolved`	After the update there were unresolved references

Example

The following code updates all the cross-references in the active document:

```
. . .
F_ApiUpdateXRefs(F_ApiGetId(0, FV_SessionId, FP_ActiveDoc),
                                 FF_XRUI_EVERYTHING);
. . .
```

F_ApiUserCancel()

`F_ApiUserCancel()` determines whether the user has chosen the Cancel command since the current callback function was called.

`F_ApiUserCancel()` is useful for clients that conduct extensive processing that the user may want to cancel. For example, if your client processes all the documents in a book, it can call `F_ApiUserCancel()` after it processes each document. If `F_ApiUserCancel()` returns `True`, your client can abort the processing.

Synopsis

```
#include "fapi.h"
. . .
IntT F_ApiUserCancel(VoidT);
```

Arguments

None.

Returns

`True` if the user has executed the Cancel gesture, or `False` if the user has not executed the Cancel gesture.

F_ApiUSleep()

`F_ApiUSleep()` . It suspends client and FrameMaker product operation for a specified number of microseconds.

IMPORTANT: *Do not call* `usleep()` *in your client. Use* `F_ApiUSleep()` *instead.*

Synopsis

```
#include "fapi.h"
. . .
Intt F_ApiUSleep(IntT microseconds);
```

Arguments

`microseconds`	The number of microseconds to suspend FrameMaker product and client operation

Returns

The number of microseconds remaining before client and FrameMaker product operation resume.

If `F_ApiUSleep()` fails, the API assigns the following value to `FA_errno`.

FA_errno value	Meaning
`FE_SystemTransport`	A transport error occurred

Example

The following code suspends client and FrameMaker product information for one tenth of a second:

```
. . .
F_ApiUSleep(100000);
. . .
```

See also

"F_ApiSleep()" on page 468.

F_ApiWinConnectSession()

F_ApiWinConnectSession() Allows you to register asynchronous clients just as you would register other clients; you can store the registration data in the client's version info file or you can pass a F_PropValsT structure to F_ApiWinConnectSession that is a list of registration data.

For more information on working with asyncronous clients see the FDK's Windows platform guide.

F_ApiWinConnectSession() is defined as:

F_ApiWinConnectSession(const F_PropValsT *connectProps, ConStringT hostname, cons struct _GUID *service):

This property	corresponds to this statement in a client's VERSIONINFO resource
FI_PLUGIN_NAME	the name of the client.
FI_PLUGIN_TYPE	the type of client.
FI_PLUGIN_PRODUCTS	the names of FrameMaker products this client supports.
FI_PLUGIN_FACET	the name of the file format supported by the client (filters, only)
FI_PLUGIN_FORMATID	a four-character string that identifies a file format (filters, only).
FI_PLUGIN_VENDOR	a four-character string that identifies the client's provider.
FI_PLUGIN_SUFFIX	the filename extension of the file type that the client filters (filters, only).
FI_PLUGIN_INFORMAT	the file format for the file to filter (filters, only)
FI_PLUGIN_OUTFORMAT	the file format for the resulting file (filters, only)
FI_PLUGIN_DESCRIPTION	a description of the client that appears when you choose About.
FI_PLUGIN_PRODUCTNAME	the name by which customers know your client.

If `connectProps` is `NULL`, the FrameMaker process uses the client's VersionInfo resource or the entries in the `.ini` file. If the client has no registration information in any of these sources, the FrameMaker process registers it as a standard client.

F_ApiWinInstallDefaultMessageFilter()

Registers the default FDK message filter for a COM session. If you have an application which initializes COM but does not install a message filter you should use F_ApiWinInstallDefaultMessageFilter() to have FrameMaker install it's default message filter.

Structured F_ApiWrapElement()

`F_ApiWrapElement()` inserts a structural element around the selected text and structural elements in a document.

If the flow that contains the selection is unstructured and the selection does not include the entire flow contents, `F_ApiWrapElement()` wraps the flow contents into a NoName element before wrapping the selection into the specified element definition.

Synopsis

```
#include "fapi.h"
. . .
VoidT F_ApiWrapElement(F_ObjHandleT docId,
    F_ObjHandleT edefId);
```

Arguments

`docId`	The ID of the document containing the selected text and elements
`edefId`	The ID of the element definition for the new element

Returns

`VoidT`.

If `F_ApiWrapElement()` fails, the API assigns one of the following values to `FA_errno`.

FA_errno value	Meaning
`FE_WrongProduct`	Current FrameMaker product isn't Structured FrameMaker
`FE_BadDocId`	Invalid document ID
`FE_BadElementDefId`	Specified element definition ID is invalid
`FE_BadSelectionForOperation`	Current text selection is invalid for this operation

Example

The following code sets the text selection and wraps the elements in it:

```
. . .
F_TextRangeT tr;
F_ObjHandleT docId, edefId;
. . .
F_ApiSetTextRange(FV_SessionId, docId, FP_TextSelection, &tr);
F_ApiWrapElement(docId, edefId);
. . .
```

See also

"F_ApiNewElement()" on page 317, and "F_ApiUnWrapElement()" on page 479.

F_Assert()

`F_Assert()` is a macro that tests whether an expression is true and calls the assertion handler if it is not.

When a client calls `F_Assert()` and assertion failure occurs, the FDK executes an assertion handler. You can use `F_SetAssert()` to specify your own assertion handler. However, after your assertion-handler function returns, the FDE's default assertion handler is called to clean up the system and exit the client properly. For more information, see "F_SetAssert()" on page 624 of the *FDK Programmer's Reference*.

Synopsis

```
#include "fdetypes.h"
#include "fassert.h"
. . .
VoidT F_Assert(BoolT p);
```

Arguments

`p`	The expression to test

Returns

`VoidT`.

Example

The following code defines and sets an assertion handler and then calls it by testing a false statement:

```
. . .
VoidT myHandler();
. . .
UCharT *nullstr = NULL;

F_SetAssert(myHandler);     /* Set assertion handler. */
F_Assert(nullstr != NULL); /* Calls assertion handler. */

VoidT myHandler()
{
   F_Printf(NULL, "Assertion failed.\n");
}
. . .
```

F_Calloc()

`F_Calloc()` allocates a memory block of `n` items of `size` bytes. It initializes the memory block to zeros.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
PtrT F_Calloc(UIntT n,
   UIntT size,
   PUCharT flags);
```

Arguments

`n`	The number of items for which to allocate memory
`size`	The size of the items
`flags`	Specifies whether to bail out (`DSE`) or return `NULL` (`NO_DSE`) if the memory you request isn't available

Returns

A pointer to the allocated memory, or `NULL` if it fails.

Example

The following code allocates memory to a pointer:

```
. . .
UCharT *ptr;

ptr = F_Calloc(128, sizeof(UCharT), NO_DSE);
if(ptr == NULL)
    F_Printf(NULL, "Couldn't Allocate memory.\n");
. . .
```

F_ChannelAppend()

`F_ChannelAppend()` appends the contents of a source channel to a destination channel. It reads the contents from the current position in the source channel and appends them to the current position in the destination channel. After `F_ChannelAppend()` has finished, the current position for both channels is at the end.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
ErrorT F_ChannelAppend(ChannelT srcChannel,
    ChannelT dstChannel);
```

Arguments

`srcChannel`	The source channel
`dstChannel`	The destination channel

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

The following code appends the contents of the file `test1.txt` to the file `test2.txt`:

```
. . .
ChannelT srcChan, dstChan;
FilePathT *path1, *path2;

path1 = F_PathNameToFilePath((StringT)"test1.txt",
                NULL, FDefaultPath);
path2 = F_PathNameToFilePath((StringT)"test2.txt",
                NULL, FDefaultPath);
if((srcChan = F_ChannelOpen(path1,"r")) == NULL) return;
if((dstChan = F_ChannelOpen(path2,"a")) == NULL) return;
F_ChannelAppend(srcChan, dstChan);
. . .
```

F_ChannelClose()

`F_ChannelClose()` performs clean-up tasks required by the operating system and frees a specified channel. You can't use a channel after you close it.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
IntT F_ChannelClose(ChannelT channel);
```

Arguments

`channel`	The channel to close

Returns

`0` if it succeeds, or a nonzero value if an error occurs.

Example

See "F_ChannelOpen()" on page 504.

F_ChannelCloseTmp()

`F_ChannelCloseTmp()` frees a temporary channel opened by `F_ChannelMakeTmp()` and deletes the temporary file associated with the channel. To avoid accumulating temporary files, use this function to close all temporary channels.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
IntT F_ChannelCloseTmp(ChannelT tmpChannel);
```

Arguments

`tmpChannel` The channel to close

Returns

`0` if it succeeds, or a nonzero value if an error occurs.

Example

See "F_ChannelMakeTmp()" on page 503.

F_ChannelEof()

`F_ChannelEof()` returns a nonzero value if the end of a specified channel has been read. If you set the file pointer to the end of the channel with `F_ChannelSeek()`, `F_ChannelEof()` returns zero.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
IntT F_ChannelEof(ChannelT channel);
```

Arguments

`channel`	The channel to check

Returns

A nonzero value if the end of a channel has been read, or zero if the end of the channel has not been read or if you have set the file pointer to the end of a channel with `F_ChannelSeek()`.

Example

See "F_ChannelRead()" on page 506.

F_ChannelFlush()

`F_ChannelFlush()` flushes all buffered output into the specified channel.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
IntT F_ChannelFlush(ChannelT channel);
```

Arguments

`channel`	The channel for which to flush output

Returns

`0` if it succeeds, or a nonzero value if an error occurs.

Example

See "F_ChannelWrite()" on page 511.

F_ChannelMakeTmp()

`F_ChannelMakeTmp()` creates and opens a temporary channel for reading and writing. The `size` flag indicates the estimated size. In the current version of the FDE, the `size` flag is not implemented. All channels are disk-based. To close a temporary channel created with `F_ChannelMakeTmp()`, use `F_ChannelCloseTmp()`.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
ChannelT F_ChannelMakeTmp(PUCharT size);
```

Arguments

`size`	Not implemented

Returns

The channel it creates, or `NULL` if it fails.

Example

The following code creates a temporary file and closes it:

```
. . .
ChannelT tmpChan;

if((tmpChan = F_ChannelMakeTmp(1024)) == NULL)
{
   F_Printf(NULL, "Couldn't open temp file.\n");
   return;
}
F_ChannelCloseTmp(tmpChan);
. . .
```

F_ChannelOpen()

`F_ChannelOpen()` opens a channel specified by a filepath.

NOTE: This function works even if `FilePathT` supplied to `F_ChannelOpen()` API is prepared from an HTTP path.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
ChannelT F_ChannelOpen(FilePathT *path,
   StringT flag);
```

Arguments

`path`	The path to open.
`flag`	The file access mode. See the table below for a list of flags.

The string `flag` can have the following values.

Flag value	Meaning
`r`	Open the channel for reading. If the channel doesn't exist, return `NULL`.
`w`	Open the channel for writing. If the specified channel already exists, destroy its contents.
`a`	Open the channel for appending. If the channel doesn't exist, create it.
`r+`	Open the channel for both reading and writing. If the channel doesn't exist, return `NULL`.
`w+`	Open the channel for both reading and writing. If the channel exists, destroy its contents.
`a+`	Open the channel for both reading and appending. If the channel doesn't exist, create it.

Returns

The opened channel if it succeeds, or `NULL` if it fails.

Example

The following code opens a channel for reading and then closes it:

```
. . .
ChannelT chan;
FilePathT *path;

path = F_PathNameToFilePath((StringT)"test.txt",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"r")) == NULL)
{
   F_Printf(NULL, "Couldn't open file.\n");
   return;
}
F_ChannelClose(chan);
. . .
```

F_ChannelPeek()

`F_ChannelPeek()` returns the next byte at the current position of a specified channel without updating the current channel position.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
IntT F_ChannelPeek(ChannelT channel);
```

Arguments

`channel`	The channel for which to get the next byte

Returns

The next byte at the current position of the channel if it succeeds, or `-1` if it has reached the end of the channel.

Example

The following code prints the sixth byte of a file:

```
. . .
ChannelT chan;
FilePathT *path;
IntT nextByte;

path = F_PathNameToFilePath((StringT)"test.txt",
                 NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"r")) == NULL) return;
if (F_ChannelSeek(chan, 5, 0)) return;
nextByte = F_ChannelPeek(chan);
if(nextByte != -1)
   F_Printf(NULL, "Next byte: %c\n", nextByte);
. . .
```

F_ChannelRead()

`F_ChannelRead()` reads from a channel into a specified memory block. If fewer than `size` bytes of data remain in the channel, `F_ChannelRead()` does not read the remaining data.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
IntT F_ChannelRead(PtrT ptr,
    IntT size,
    IntT nitems,
    ChannelT channel);
```

Arguments

`ptr`	The memory block to which to copy the contents of the channel
`size`	The size of the items to read
`nitems`	The number of items to read
`channel`	The channel from which to read the items

Returns

The number of items read.

Example

The following code reads text from a file and prints it:

```
. . .
#define BUFFERSIZE (IntT)257
ChannelT chan;
FilePathT *path;
UCharT ptr[BUFFERSIZE];
IntT numread;

path = F_PathNameToFilePath((StringT)"test.txt",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"r")) == NULL)
   return;
while(!F_ChannelEof(chan))
{
   numread = F_ChannelRead(ptr, sizeof(UCharT),
                       BUFFERSIZE-1, chan);
   ptr[numread] = '\0';
   F_Printf(NULL, "%s\n", (StringT)ptr);
}
. . .
```

F_ChannelSeek()

`F_ChannelSeek()` sets the position for the next input operation on an input channel.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
IntT F_ChannelSeek(ChannelT channel,
   NativeLongT offset,
   PUCharT mode);
```

Arguments

`channel`	The channel for which to set the input position.
`offset`	The offset at which to set the input position. It is relative to the position specified by `mode`.
`mode`	Specifies what the offset is relative to. See the table below for a list of values.

The `mode` parameter can have the following values.

Mode value	Meaning
`0`	Offset is relative to the beginning of the channel
`1`	Offset is relative to the current position of the channel
`2`	Offset is relative to the end of the channel

Returns

`0`, if it succeeds or `-1` if the seek is illegal.

Example

See "F_ChannelPeek()" on page 506.

F_ChannelSize()

`F_ChannelSize()` returns the size in bytes of the file specified by a channel. It is useful for keeping track of how much you have written to a channel.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
NativeLongT F_ChannelSize(ChannelT channel);
```

Arguments

`channel`	The channel to check

Returns

The size of the specified channel in bytes.

Example

The following code prints the size of a channel:

```
. . .
ChannelT chan;
FilePathT *path;
path = F_PathNameToFilePath((StringT)"test.txt",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"r")) == NULL) return;
F_Printf(NULL, "Channel size: %d bytes\n", F_ChannelSize(chan));
F_ChannelClose(chan);
. . .
```

F_ChannelTell()

`F_ChannelTell()` determines the offset of the current byte relative to the beginning of the specified input channel.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
NativeLongT F_ChannelTell(ChannelT channel);
```

Arguments

`channel`	The channel to check

Returns

The offset of the current byte relative to the beginning of the channel.

Example

The following code prints the current offset of a channel:

```
. . .
ChannelT chan;
FilePathT *path;

path = F_PathNameToFilePath((StringT)"test.txt",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"r")) == NULL)
{
   F_Printf(NULL, "Couldn't open file.\n");
   return;
}
F_Printf(NULL, "Current offset: %d bytes\n",
       F_ChannelTell(chan));
. . .
```

F_ChannelWrite()

`F_ChannelWrite()` writes data from a specified memory block to a channel. To make sure the data is written from the internal buffer to the channel, call `F_ChannelFlush()` or `F_ChannelClose()`.

Synopsis

```
#include "fdetypes.h"
#include "fchannel.h"
. . .
IntT F_ChannelWrite(PtrT ptr,
   IntT size,
   IntT nitems,
   ChannelT channel);
```

Arguments

ptr	The memory block containing the items to write to the channel
size	The size of the items to write
nitems	The number of items to write
channel	The channel to which to write the items

Returns

The number of items written to the channel.

Example

The following code writes text to a file:

```
. . .
ChannelT chan;
FilePathT *path;
static UCharT ptr[] = "All experience hath shown";

path = F_PathNameToFilePath((StringT)"test.txt",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"w")) == NULL)
   return;
F_ChannelWrite(ptr, sizeof(UCharT), F_StrLen(ptr), chan);
F_ChannelFlush(chan);
. . .
```

F_CharIsAlphabetic()

F_CharIsAlphabetic() determines whether a specified character is an alphabetic character in the FrameMaker product character set.

NOTE: This function is designed to work with the FrameMaker encoding for Roman characters. It will also work with single-byte characters in the range x00 through x7F for Asian language encodings.

Synopsis

```
#include "fdetypes.h"
#include "fcharmap.h"
. . .
BoolT F_CharIsAlphabetic(UCharT c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is alphabetic, or `0` if it isn't.

Example

The following code tests the character `3` to determine whether it is alphabetic. It prints the string `3 is not alphabetic`.

```
. . .
if(!F_CharIsAlphabetic((UCharT) '3'))
   F_Printf(NULL, "3 is not alphabetic\n");
. . .
```

F_CharIsAlphaUTF8()

Determines whether a specified UTF-8 character is an alphabetic character

Synopsis

```
#include "fencode.h"

. . .

BoolT F_CharIsAlphaUTF8 (const UCharT *c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is an alphabetic character, or 0 if it isn't

Example

The following code prints Δ is Alphabetic

```
#include "fencode.h"
. . .
UCharT upper_delta[]={0xCE, 0x94};
F_FdeInitFontEncs((ConStringT)"UTF-8");
if (F_CharIsAlphaUTF8(upper_delta))
   F_Printf(NULL,"%C is Alphabetic", upper_delta);
```

F_CharIsAlphaNumeric()

`F_CharIsAlphaNumeric()` determines whether a specified character is an alphanumeric (0, 1...9, a...z, or A...Z) character in the FrameMaker product character set.

NOTE: This function is designed to work with the FrameMaker encoding for Roman characters. It will also work with single-byte characters in the range x00 through x7F for Asian language encodings.

Synopsis

```
#include "fdetypes.h"
#include "fcharmap.h"
. . .
BoolT F_CharIsAlphaNumeric(UCharT c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is an alphanumeric character, or `0` if it isn't.

Example

The following code tests the character `2` to determine whether it is alphanumeric. It prints the string `2 is alphanumeric`.

```
. . .
if(F_CharIsAlphaNumeric((UCharT)'2'))
   F_Printf(NULL, "2 is alphanumeric\n");
. . .
```

F_CharIsAlphaNumericUTF8()

Determines whether a specified UTF-8 character is an alphanumeric character

NOTE: This function doesn't return True for numeric characters like Ⅳ (code point 0x2163) that aren't in a decimal system.

Synopsis

```
#include "fencode.h"

. . .

BoolT F_CharIsAlphaNumericUTF8 (const UCharT *c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is an alphanumeric character, or 0 if it isn't

Example

The following code prints Δ is AlphaNumeric

```
#include "fencode.h"
. . .
UCharT upper_delta[]={0xCE, 0x94};
F_FdeInitFontEncs((ConStringT)"UTF-8");
if (F_CharIsAlphaNumericUTF8(upper_delta))
   F_Printf(NULL,"%C is AlphaNumeric", upper_delta);
...
```

F_CharIsControl()

`F_CharIsControl()` determines whether a specified character is a FrameMaker product control character. It returns `True` for ASCII decimal values 8 (tab), 9 (forced return), 10 (end-of-paragraph), and 11 (end-of-flow).

NOTE: This function is designed to work with the FrameMaker encoding for Roman characters. It will also work with single-byte characters in the range x00 through x7F for Asian language encodings.

Synopsis

```
#include "fdetypes.h"
#include "fcharmap.h"
. . .
BoolT F_CharIsControl(UCharT c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is a FrameMaker product control character, or `0` if it isn't.

Example

The following code tests the character `8` (octal 10) to determine whether it is a control character. It prints the string `The character is a control character`.

```
. . .
if(F_CharIsControl((UCharT)'\10'))
   F_Printf(NULL, "The character is a control character\n");
. . .
```

F_CharIsControlUTF8()

Determines whether a specified character is a FrameMaker control character. It returns True for decimal values 8 (tab), 9 (forced return), 10 (end-of-paragraph), and 11 (end-of-flow).

Synopsis

```
#include "fencode.h"

. . .

BoolT F_CharIsAlphaUTF8 (const UCharT *c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is a FrameMaker control character, or 0 if it isn't

Example

The following code prints The character is a control character

```
#include "fencode.h"
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
if (F_CharIsEolUTF8(&"\x07"))/* decimal value of 0x07 is 7*/
   F_Printf(NULL,"The character is a control character");
...
```

F_CharIsDoubleByte()

`F_CharIsDoubleByte()` determines whether a pair of `UCharT`s form a valid double-byte character in the specified encoding.

Synopsis

```
#include "fencode.h"
. . .
BoolT F_CharIsDoubleByte(
   UCharT firstChar,
   UCharT firstChar,
   FontEncIdT feId);
```

Arguments

`firstChar`	The first character to check
`secondChar`	The second character to check
`feId`	The ID of the font encoding to check against

Returns

A nonzero value if `firstChar` and `secondChar` comprise a double-byte character, or `0` if not.

Example

The following code scans a string for a double-byte Japanese character.

```
. . .
StringT myString;
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
IntT i;
BoolT foundIt;
. . .
/* Get myString */
. . .
i = 1;
foundIt = False;
while (myString) {
    if(F_CharIsDoubleByte(myString[i], myString[i+1], feId) {
        foundIt = True;
        break;
    }
    i++;
}
if (foundIt) {
/* The UCharT's myString[i] and myString[i+1] comprise a */
/* double-byte char, so you can process them as such. */
. . .
}
. . .
```

F_CharIsDoubleByteFirst()

`F_CharIsDoubleByteFirst()` determines whether a `UCharT` is identified in the specified font encoding as the first byte of a double-byte character.

Synopsis

```
#include "fencode.h"
. . .
BoolT F_CharIsDoubleByteFirst(UCharT c, FontEncIdT feId);
```

Arguments

`c`	The character to check
`feId`	The ID of the font encoding to check against

Returns

A nonzero value if `c` is the first byte of a double-byte character, or `0` if it isn't.

Example

The following code scans a string for a first byte of a Japanese character.

```
. . .
StringT myString;
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
IntT i;
BoolT foundFirst;
. . .
/* Get myString */
. . .
i = 1;
foundFirst = False;
while (myString) {
    if(F_CharIsDoubleByteFirst(myString[i], feId) {
        foundFirst = True;
        break;
    }
    i++;
}
if (foundFirst) {
    /* The UCharT myString[i] is the first of a double-byte char
*/
. . .
}
. . .
```

F_CharIsDoubleByteSecond()

`F_CharIsDoubleByteSecond()` determines whether a `UCharT` is identified in the specified font encoding as the second byte of a double-byte character.

Synopsis

```
#include "fencode.h"
. . .
BoolT F_CharIsDoubleByteSecond(UCharT c, FontEncIdT feId);
```

Arguments

`c`	The character to check
`feId`	The ID of the font encoding to check against

Returns

A nonzero value if the `c` is the second byte of a double-byte character, or `0` if it isn't.

Example

The following code scans a string for a first byte of a Japanese character, and then checks whether the next `UCharT` is a second byte of a double-byte character. If not, it traps the character as belonging to the single-byte range of a Japanese font.

```
. . .
StringT myString;
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
IntT i;
BoolT foundSingle;
. . .
/* Get myString */
. . .
i = 1;
foundSingle = False;
while (myString) {
    if(F_CharIsDoubleByteFirst(myString[i], feId) &&
        ! F_CharIsDoubleByteSecond(myString[i + 1], feId)) {
        foundSingle = True;
        break;
    }
    i++;
}
if (foundSingle) {
/* The UCharT myString[i] is in the single-byte range of a */
/* Japanese font */
. . .
}
. . .
```

F_CharIsEol()

`F_CharIsEol()` determines whether a specified character is a FrameMaker product end-of-line (EOL) character. It returns `True` for ASCII decimal values 9 (forced return), 10 (end-of-paragraph), and 11 (end-of-flow).

NOTE: This function is designed to work with the FrameMaker encoding for Roman characters. It will also work with single-byte characters in the range x00 through x7F for Asian language encodings.

Synopsis

```
#include "fdetypes.h"
#include "fcharmap.h"
. . .
BoolT F_CharIsEol(UCharT c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is a FrameMaker product EOL character, or `0` if it isn't.

Example

The following code tests the character `9` (octal 11) to determine whether it is an EOL character. It prints the string `The character is an EOL character`.

```
. . .
if(F_CharIsEol((UCharT) '\11'))
    F_Printf(NULL, "The character is an EOL character\n");
. . .
```

F_CharIsEolUTF8()

Determines whether a specified character is a FrameMaker end-of-line (EOL) character. It returns `True` for decimal values `9` (forced return), `10` (end-of-paragraph), and `11` (end-of-flow).

Synopsis

```
#include "fencode.h"

. . .

BoolT F_CharIsEolUTF8 (const UCharT *c);
```

Arguments

Returns

A nonzero value if the character is a FrameMaker end-of-line (EOL) character, or 0 if it isn't

Example

The following code prints The character is an EOL character

```
#include "fencode.h"
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
if (F_CharIsEolUTF8(&"\x0B"))/* decimal value of 0x0B is 11 */
   F_Printf(NULL,"The character is an EOL character);
...
```

F_CharIsHexadecimal()

`F_CharIsHexadecimal()` determines whether a specified character is a hexadecimal digit (0, 1...9, a...f, or A...F).

NOTE: This function is designed to work with the FrameMaker encoding for Roman characters. It will also work with single-byte characters in the range x00 through x7F for Asian language encodings.

Synopsis

```
#include "fdetypes.h"
#include "fcharmap.h"
. . .
BoolT F_CharIsHexadecimal(UCharT c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is a hexadecimal character, or `0` if it isn't.

Example

The following code tests the character `f` to determine whether it is hexadecimal. It prints the string `The character is a Hex digit`.

```
. . .
if(F_CharIsHexadecimal((UCharT) 'f'))
   F_Printf(NULL, "The character is a Hex digit.\n");
. . .
```

F_CharIsHexadecimalUTF8()

Determines whether a specified UTF-8 character is a hexadecimal digit (0, 1...9, a...f, or A...F)

Synopsis

```
#include "fencode.h"

. . .

BoolT F_CharIsHexadecimalUTF8 (const UCharT *c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is an alphabetic character, or 0 if it isn't

Example

The following code prints `F` is Hexadecimal

```
#include "fencode.h"
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
if (F_CharIsHexadecimalUTF8(&"F"))
   F_Printf(NULL,"F is Hexadecimal");
...
```

F_CharIsLower()

`F_CharIsLower()` determines whether a specified character is a lowercase character in the FrameMaker product character set.

NOTE: This function is designed to work with the FrameMaker encoding for Roman characters. It will also work with single-byte characters in the range x00 through x7F for Asian language encodings.

Synopsis

```
#include "fdetypes.h"
#include "fcharmap.h"
. . .
BoolT F_CharIsLower(UCharT c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is a lowercase character, or `0` if it isn't.

Example

The following code tests the character `a` to determine whether it is lowercase. It prints the string `The character is lowercase`.

```
. . .
if(F_CharIsLower((UCharT) 'a'))
   F_Printf(NULL, "The character is lowercase\n");
. . .
```

F_CharIsLowerUTF8()

Determines whether a specified UTF-8 character is a lowercase character

Synopsis

```
#include "fencode.h"

. . .

BoolT F_CharIsLowerUTF8 (const UCharT *c);
```

Arguments

c	The character to check.

Returns

A nonzero value if the character is a lowercase character, or 0 if it isn't

Example

The following code prints δ is in Lowercase

```
#include "fencode.h"
. . .
UCharT lower[] = {0xCE, 0xB4};
F_FdeInitFontEncs((ConStringT)"UTF-8");
if (F_CharIsLowerUTF8(lower))
F_Printf(NULL,"%C is in Lowercase", lower);
...
```

F_CharIsNumeric()

`F_CharIsNumeric()` determines whether a specified character is a numeric character in the FrameMaker product character set.

NOTE: This function is designed to work with the FrameMaker encoding for Roman characters. It will also work with single-byte characters in the range x00 through x7F for Asian language encodings.

Synopsis

```
#include "fdetypes.h"
#include "fcharmap.h"
. . .
BoolT F_CharIsNumeric(UCharT c);
```

Arguments

c	The character to check

Returns

A nonzero value if the character is a numeric character, or `0` if it isn't.

Example

The following code tests the character `8` to determine whether it is numeric. It prints the string `The character is numeric`.

```
. . .
if(F_CharIsNumeric((UCharT) '8'))
   F_Printf(NULL, "The character is numeric\n");
. . .
```

F_CharIsNumericUTF8()

Determines whether a specified UTF-8 character is a numeric character in a decimal system.

NOTE: This function doesn't return True for numeric characters like Ⅳ (code point 0x2163) that aren't in a decimal system.

Synopsis

```
#include "fencode.h"

. . .

BoolT F_CharIsNumericUTF8 (const UCharT *c);
```

Arguments

Returns

A nonzero value if the character is a numeric character in a decimal system, or 0 if it isn't

Example

The following code prints ४ is Numeric

```
#include "fencode.h"
. . .
StringT devanagiri_four="\xE0\xA5\xAA";
F_FdeInitFontEncs((ConStringT)"UTF-8");
if (F_CharIsNumericUTF8(devanagiri_four))
   F_Printf(NULL,"%C is Numeric", devanagiri_four);
...
```

F_CharIsUpper()

`F_CharIsUpper()` determines whether a specified character is an uppercase character in the FrameMaker product character set.

NOTE: This function is designed to work with the FrameMaker encoding for Roman characters. It will also work with single-byte characters in the range x00 through x7F for Asian language encodings.

Synopsis

```
#include "fdetypes.h"
#include "fcharmap.h"
. . .
BoolT F_CharIsUpper(UCharT c);
```

Arguments

`c`	The character to check

Returns

A nonzero value if the character is an uppercase character, or `0` if it isn't.

Example

The following code tests the character `H` to determine whether it is uppercase. It prints the string `The character is uppercase`.

```
. . .
if(F_CharIsUpper((UCharT) 'H'))
    F_Printf(NULL, "The character is uppercase\n");
. . .
```

F_CharIsUpperUTF8()

Determines whether a specified UTF-8 character is an uppercase character

Synopsis

```
#include "fencode.h"

. . .

BoolT F_CharIsUpperUTF8 (const UCharT *c);
```

Arguments

c	The character to check

Returns

A nonzero value if the character is an uppercase character, or 0 if it isn't

Example

The following code prints Δ is in Uppercase

```
#include "fencode.h"
. . .
UCharT upper[]={0xCE, 0x94};
F_FdeInitFontEncs((ConStringT)"UTF-8");
if (F_CharIsUpperUTF8(upper))
   F_Printf(NULL,"%C is in Uppercase", upper);
...
```

F_CharToLower()

`F_CharToLower()` converts a character in the FrameMaker product character set to lowercase.

NOTE: This function is designed to work with the FrameMaker encoding for Roman characters. It will also work with single-byte characters in the range x00 through x7F for Asian language encodings.

Synopsis

```
#include "fdetypes.h"
#include "fcharmap.h"
. . .
UCharT F_CharToLower(UCharT c);
```

Arguments

c	The character to convert

Returns

The specified character, converted to lowercase.

Example

The following code converts a mixed-case string to lowercase. It prints the string `mixed case`.

```
. . .
StringT s;
IntT i;

s = F_StrCopyString((StringT) "mIxeD Case");
for (i= F_StrLen(s); i--; i>= 0)
   s[i] = F_CharToLower(s[i]);
F_Printf(NULL, "%s\n", s);
. . .
```

F_CharToLowerUTF8()

Converts a UTF-8 character to lowercase

Synopsis

```
#include "fencode.h"

. . .

VoidT F_CharToLowerUTF8(const UCharT *c, UCharT *newchar);
```

Arguments

c	The character to convert
`newchar`	Pointer to the converted character. It must point to a block of modifiable memory large enough to store the converted character (a UTF-8 character requires at most 4 bytes).

Returns

`VoidT`

Example

The following code prints Lowercase of Δ is δ

```
#include "fencode.h"
. . .
UCharT upper[]={0xCE, 0x94};
UCharT lower[4];
F_FdeInitFontEncs((ConStringT)"UTF-8");
F_CharToLowerUTF8(upper, lower);
F_Printf(NULL,"Lowercase of %C is %C", upper, lower);
...
```

F_CharToUpper()

`F_CharToUpper()` converts a character in the FrameMaker product character set to uppercase.

NOTE: This function is designed to work with the FrameMaker encoding for Roman characters. It will also work with single-byte characters in the range x00 through x7F for Asian language encodings.

Synopsis

```
#include "fdetypes.h"
#include "fcharmap.h"
. . .
UCharT F_CharToUpper(UCharT c);
```

Arguments

`c`	The character to convert

Returns

The specified character, converted to uppercase.

Example

The following code converts a mixed-case string to uppercase. It prints the string `MIXED CASE`.

```
. . .
StringT s;
IntT i;

s = F_StrCopyString((StringT) "mIxeD Case");
for (i=0; i < F_StrLen(s); i++)
   s[i] = F_CharToUpper(s[i]);
F_Printf(NULL, "%s\n", s);
. . .
```

F_CharToUpperUTF8()

Converts a UTF-8 character to uppercase

Synopsis

```
#include "fencode.h"

. . .

VoidT F_CharToUpperUTF8(const UCharT *c, UCharT *newchar);
```

Arguments

`c`	The character to convert
`newchar`	Pointer to the converted character. It must point to a block of modifiable memory large enough to store the converted character (a UTF-8 character requires at most 4 bytes).

Returns

`VoidT`

Example

The following code prints Uppercase of δ is Δ

```
#include "fencode.h"
. . .
UCharT upper[4];
UCharT lower[] = {0xCE, 0xB4};
F_FdeInitFontEncs((ConStringT)"UTF-8");
F_CharToUpperUTF8(lower, upper);
F_Printf(NULL,"Uppercase of %C is %C", lower, upper);
...
```

F_CharUTF16ToUTF8()

Converts a UTF-16 character to a UTF-8 character

Synopsis

```
#include "fencode.h"
. . .
IntT F_CharUTF16ToUTF8 (const UChar16T *src, UCharT *dest);
```

Arguments

src	The character to convert
dest	Pointer to the converted character. It must point to a block of modifiable memory large enough to store the converted character (a UTF-8 character requires at most 4 bytes).

Returns

The number of bytes (or UTF-8 code points) written in dest

Example

The following code prints `1Б2о3г`

```
#include "fencode.h"
. . .
UChar16T russian_U16[]={0x0411, 0x043E, 0x0433, 0x0000};
UChar16T *t = russian_U16;
UCharT dest[5];
IntT i;

F_FdeInitFontEncs((ConStringT)"UTF-8");
while(*t)
   {
   F_CharUTF16ToUTF8(t,dest);
   F_Printf("%d%C",i,dest); /* dest is not null terminated
here*/
   F_UTF16NextChar(&t);
   }
...
```

F_CharUTF32ToUTF8()

Converts a UTF-32 character to a UTF-8 character

Synopsis

```
#include "fencode.h"
. . .
IntT F_CharUTF32ToUTF8 (UChar32T src, UCharT *dest);
```

Arguments

src	The character to convert
dest	Pointer to the converted character. It must point to a block of modifiable memory large enough to store the converted character (a UTF-8 character requires at most 4 bytes).

Returns

The number of bytes (or UTF-8 code points) written in dest

Example

The following code prints 1Б2о3г

```
#include "fencode.h"
. . .
UChar32T russian_U32[]={0x0411, 0x043E, 0x0433, 0x0000};
UCharT dest[5];
IntT i;
F_FdeInitFontEncs((ConStringT)"UTF-8");
for(i=0;i<3;i++)
    {
    dest[F_CharUTF32ToUTF8(russian_U32[i],dest)]=0;
    F_Printf("%d%s",i,dest); /* note dest is null terminated here
    */
    }
...
```

F_CharUTF8ToUTF16()

Converts a UTF-8 character to a UTF-16 character

Synopsis

```
#include "fencode.h"
. . .
IntT F_CharUTF8ToUTF16 (const UCharT *src, UChar16T *dest)
```

Arguments

src	The character to convert
dest	Pointer to the converted character. It must point to a block of modifiable memory large enough to store the converted character (a UTF-16 character requires at most 2 code units of 2 bytes each).

Returns

The number of UChar16T (or UTF-16 code points) written in dest

Example

The following code prints 1,0x411

```
#include "fencode.h"
. . .
UChar16T russian_U16[2];
IntT n;
F_FdeInitFontEncs((ConStringT)"UTF-8");

n = F_CharUTF8ToUTF16("\xD0\x91",russian_U16);
F_Printf("%d,%x", n, russian_U16[0]);
...
```

F_CharUTF8ToUTF32()

Converts a UTF-8 character to a UTF-32 character

Synopsis

```
#include "fencode.h"
. . .
UChar32T F_CharUTF8ToUTF32 (const UCharT *src);
```

Arguments

`src`	The character to convert

Returns

The character in UTF-32

Example

The following code prints 0x411

```
#include "fencode.h"
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
F_Printf("%x",F_CharUTF8ToUTF32("\xD0\x91"));
...
```

F_ClearHandle()

`F_ClearHandle()` initializes to zero a handle's block of data.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
ErrorT F_ClearHandle(HandleT handle);
```

Arguments

`handle`	The handle specifying the block of data to initialize

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

See "F_AllocHandle()" on page 69.

F_ClearPtr()

`F_ClearPtr()` initializes the memory block associated with a pointer to zero.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
ErrorT F_ClearPtr(PtrT ptr,
   UIntT size);
```

Arguments

`ptr`	The pointer specifying the block of data to initialize
`size`	The size of the block of data

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

See "F_Alloc()" on page 68.

F_CopyPtr()

`F_CopyPtr()` copies from the memory area specified by one pointer to the memory area specified by another pointer.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
ErrorT F_CopyPtr(PtrT srcPtr,
    PtrT dstPtr,
    UIntT numBytes);
```

Arguments

`srcPtr`	The pointer to the memory to be copied. It must be a valid pointer pointing to at least `numBytes` of memory.
`dstPtr`	The pointer to the memory to copy over. It must be a valid pointer pointing to at least `numBytes` of modifiable memory.
`numBytes`	The number of bytes to be copied.

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

The following code copies from the memory area specified by one pointer to the memory area specified by another pointer:

```
. . .
UCharT *srcPtr = NULL, *dstPtr = NULL;

srcPtr = F_Alloc(256, NO_DSE);
if(srcPtr == NULL)
   return;
dstPtr = F_Alloc(256, NO_DSE);
if(dstPtr == NULL)
   return;
if (F_CopyPtr(srcPtr,  dstPtr, 256) != FdeSuccess)
   F_Printf(NULL, "Couldn't copy pointer.\n");
. . .
```

F_DeleteFile()

`F_DeleteFile()` removes the file or directory specified by a filepath. `F_DeleteFile()` fails if permission is denied, the filepath specifies a file or directory that does not exist, or the filepath specifies a directory that is not empty.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
ErrorT F_DeleteFile(FilePathT *filepath);
```

Arguments

`filepath`	The filepath of the file to delete

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

The following code deletes the file `tmp.txt` from the `tmp` directory:

```
. . .
FilePathT *path;

path = F_PathNameToFilePath("\<r\>\<c\>tmp\<c\>tmp.txt",
                NULL, FDIPath);
F_DeleteFile(path);
. . .
```

F_DigitValue()

Returns the numerical value of a UTF-8 character that is a decimal numeric

NOTE: This function doesn't return the value of numeric characters like Ⅳ (code point 0x2163) that aren't in a decimal system.

Synopsis

```
#include "fencode.h"

. . .

IntT F_DigitValue (UCharT *s);
```

Arguments

`s`	The character whose numeric value must be determined.

Returns

The numeric value (0-9) of the character if it is numeric, or -1 if it isn't

Example

The following code prints ४ + २ = 6

```
...
StringT devanagiri_four="\xE0\xA5\xAA";
StringT devanagiri_two ="\xE0\xA5\xA8";
IntT res;
F_FdeInitFontEncs((ConStringT)"UTF-8");
res = F_DigitValue(devanagiri_four)
+F_DigitValue(devanagiri_two);
F_Printf(NULL,"%C + %C is %d", devanagiri_four, devanagiri_two,
res);
...
```

F_DuplicateHandle()

`F_DuplicateHandle()` copies the block of memory specified by a handle to another handle.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
BoolT F_DuplicateHandle(HandleT srcHandle,
   HandleT dstHandle);
```

Arguments

`srcHandle`	A handle specifying the block of memory to be copied
`dstHandle`	A handle specifying the block of memory to which to copy

If the block of memory specified by `srcHandle` is larger than that specified by `dstHandle`, `F_DuplicateHandle()` copies only as much as `dstHandle` will hold. It does not allocate additional memory to `dstHandle`.

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

The following code duplicates a handle:

```
. . .
HandleT srcHndl, dstHndl;

srcHndl = F_AllocHandle(66000, NO_DSE);
dstHndl = F_AllocHandle(66000, NO_DSE);
F_DuplicateHandle(srcHndl, dstHndl);
. . .
```

F_DuplicatePtr()

`F_DuplicatePtr()` allocates a new block of memory and copies the contents of a specified block of memory to it. `F_DuplicatePtr()` doesn't perform a clever copy. If the specified block of memory contains pointers, it duplicates the pointers but not the blocks to which they point.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
PtrT F_DuplicatePtr(PtrT srcPtr,
   UIntT size,
   PUCharT flags);
```

Arguments

`srcPtr`	A pointer specifying the block of memory to duplicate
`size`	The size of the block of memory to duplicate
`flags`	Flag specifying whether to bail out (`DSE`) or return `NULL` (`NO_DSE`) if the memory you request isn't available

Returns

The new block of memory if it succeeds, or `NULL` if it fails.

Example

The following code duplicates a pointer:

```
. . .
UCharT *srcPtr = NULL, *dstPtr = NULL;

srcPtr = F_Alloc(256, NO_DSE);
if(srcPtr == NULL) return;
dstPtr = F_DuplicatePtr(srcPtr,  256, NO_DSE);
if(dstPtr == NULL)
   F_Printf(NULL, "Couldn't duplicate pointer.\n");
. . .
```

F_Exit()

`F_Exit()` cleans up and exits the application with number `n`.

IMPORTANT: *If you are writing an API client, do not call* `F_Exit()`*. Use* `F_ApiBailOut()` *instead.*

Synopsis

```
#include "fdetypes.h"
#include "fprogs.h"
. . .
VoidT F_Exit(IntT n);
```

Arguments

`n`	The number with which to exit

Returns

`VoidT`.

Example

The following code exits an application:

```
. . .
F_Exit(-5);
. . .
```

F_FdeEncodingsInitialized()

Determines whether the FDE encoding data has been correctly initialized. If this doesn't return true, `F_FdeInit` or `F_FdeInitFontEncs` has not been called or has failed and the FDE won't function correctly. Use this function, along with `F_GetICUDataDir`, to check that the FDE has been correctly set up before making FDE calls.

Synopsis

```
#include "fencode.h"

. . .

BoolT F_FdeEncodingsInitialized (VoidT);
```

Arguments

`VoidT`

Returns

A nonzero value if FDE font encoding has been initialized by `F_FdeInit` or `F_FdeInitFontEncs`, or `0` if it hasn't.

Example

The following code prints `Font Encoding Data Initialized`

```
#include "fencode.h"
...
F_FdeInitFontEncs((ConStringT)"UTF-8");
if (F_FdeEncodingsInitialized())
F_Printf(NULL,"Font Encoding Data Initialized");
...
```

F_FdeInit()

`F_FdeInit()` initializes the FDE. Call it before you call any other FDE functions.

IMPORTANT: *If you call* `F_ApiBailOut()`*, you must call* `F_FdeInit()` *to reinitialize the FDE.*

Synopsis

```
#include "fdetypes.h"
. . .
ErrorT F_FdeInit(VoidT);
```

Returns

`VoidT`.

Example

The following code initializes the FDE:

```
. . .
F_FdeInit();
. . .
```

F_FdeInitFontEncs()

`F_FdeInitFontEncs()` initializes the current session's font encoding data for your client to use. It also sets the font encoding for any dialog boxes your client displays. Call it before you call any other FDE functions that have to do with font encodings.

IMPORTANT: *If you call* `F_ApiBailOut()`*, you must call* `F_FdeInitFontEncs()` *to reinitialize the font encoding data for your client.*

Calling this function with "`UTF-8`" as a parameter enables *Unicode Mode* for the FDE. Calling this function with any other encoding disables *Unicode Mode* and enables *Compatibility Mode* for the FDE.

Synopsis

```
#include "fencode.h"
. . .
FontEncIdT F_FdeInitFontEncs(ConStringT fontEncName);
```

Arguments

`fontEncName`	The name of the font encoding to use for your client's dialog boxes.

Possible values for `fontEncName` are:

Value	Means
`FrameRoman`	Roman text
`JISX0208.ShiftJIS`	Japanese text
`BIG5`	Traditional Chinese text
`GB2312-80.EUC`	Simplified Chinese text
`KSC5601-1992`	Korean text
`UTF-8`	Enables *Unicode Mode* for the FDE

Returns

The ID of the font encoding you assigned to your dialog boxes. If the FDE does not recognize `fontEncName` as a valid encoding name for the current session, this function returns the ID for the Roman encoding.

Example

The following code initializes the FDE and ensures the dialog box encoding is one the client can support. If the dialog box encoding for the current session is Japanese or Simplified Chinese, it passes that encoding the `F_FdeInitFontEncs()`. Otherwise, it passes Roman to `F_FdeInitFontEncs()`:

```
. . .
FontEncIdT feId;
StringT encName;

F_FdeInit();
encName = F_ApiGetString(0, FV_SessionId,
FP_DialogEncodingName);
if (F_StrIEqual( encName, "JISX0208.ShiftJIS") ||
                F_StrIEqual( encName, "GB2312-80")
   feId = F_FdeInitFontEncs((ConStringT) encName);
else
   feId = F_FdeInitFontEncs((ConStringT) "FrameRoman");
. . .
```

F_FilePathBaseName()

`F_FilePathBaseName()` returns the *basename* of a specified filepath. If a filepath ends with a directory delimiter, its basename is the string immediately before the directory delimiter. If a filepath doesn't end with a directory delimiter, the basename is the string after the last directory delimiter. For example, the basename of the following filepaths:

```
/usr/root/mydirectory/
/usr/root/mydirectory
```

is `mydirectory`.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
StringT F_FilePathBaseName(FilePathT *filepath);
```

Arguments

`filepath`	The filepath for which to return the basename

Returns

The basename of the specified filepath, or `NULL` if there is insufficient memory. `F_FilePathBaseName()` returns an empty string if `filepath` specifies a root directory. It returns `NULL` if there is not enough memory. `F_FilePathBaseName()` allocates the returned string with `F_Alloc()`. You must use `F_Free()` to free this string.

Example

The following code prints the string `Basename: tmp.txt`:

```
. . .
FilePathT *path;
StringT basename;

path = F_PathNameToFilePath("\<r\>\<c\>tmp\<c\>tmp.txt",
                 NULL, FDIPath);
basename = F_FilePathBaseName(path);
F_Printf(NULL, "Basename: %s\n", basename);
F_Free(basename);
. . .
```

F_FilePathCloseDir()

`F_FilePathCloseDir()` closes a file handle opened by `F_FilePathOpenDir()`. After the handle is closed, you can't reference it.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
VoidT F_FilePathCloseDir(DirHandleT handle);
```

Arguments

`handle`	The file handle to close

Returns

`VoidT`.

Example

For an example of `F_FilePathCloseDir()` in use, see "F_FilePathGetNext()" on page 550.

F_FilePathCopy()

`F_FilePathCopy()` returns a copy of a specified filepath.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
FilePathT *F_FilePathCopy(FilePathT *filepath);
```

Arguments

`filepath`	The filepath to return a copy of

Returns

A copy of the specified filepath, or `NULL` if it fails to allocate enough memory for the new filepath.

Example

The following code copies a filepath:

```
. . .
FilePathT *path1, *path2;

path1 = F_PathNameToFilePath("\<r\>\<c\>tmp\<c\>tmp.txt",
                NULL, FDIPath);
path2 = F_FilePathCopy(path1);
. . .
F_FilePathFree(path1);
F_FilePathFree(path2);
. . .
```

F_FilePathFree()

`F_FilePathFree()` frees the resources associated with a filepath.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
ErrorT F_FilePathFree(FilePathT *path);
```

Arguments

`path`	The filepath to free

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

See “F_FilePathCopy()” on page 548.

F_FilePathGetNext()

`F_FilePathGetNext()` returns the next file in a specified directory. You can use it to scan the files in a directory. To get the directory handle, use `F_FilePathOpenDir()`. Each time you call `F_FilePathGetNext()`, it allocates a new `FilePathT` structure. Use `F_FilePathFree()` to free each `FilePathT` structure when you are done with it.

It is illegal to call `F_FilePathGetNext()` if you have closed the specified directory by calling `F_FilePathCloseDir()`.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
FilePathT *F_FilePathGetNext(DirHandleT handle, IntT *statusp);
```

Arguments

`handle`	The directory for which to get the next file
`statusp`	A pointer to memory in which the function can store a status value

Returns

The next file or subdirectory in the specified directory, or `NULL` if there is no file or subdirectory entry left in the directory. If `F_FilePathGetNext()` fails to construct the filepath, it returns `NULL` and sets the value of `statusp` to a nonzero value.

Example

The following code prints the pathnames of all the files and directories in the `tmp` directory:

```
. . .
DirHandleT handle;
FilePathT *path, *file;
IntT statusp;
StringT pathname;

/* Get filepath and directory handle of tmp directory. */
path = F_PathNameToFilePath((StringT)"\<r\>\<c\>tmp",
                    NULL, FDIPath);
handle = F_FilePathOpenDir(path, &statusp);
if (handle == 0)
{
   F_FilePathFree(path);
   return;
}

/* Traverse files and directories in tmp directory. */
while((file = F_FilePathGetNext(handle, &statusp)) != NULL)
{
   pathname = F_FilePathToPathName(file, FDIPath);
   F_Printf(NULL, "%s\n", pathname);
   F_Free(pathname);
   F_FilePathFree(file);
}

/* Free tmp directory handle and filepath. */
F_FilePathCloseDir(handle);
F_FilePathFree(path);
. . .
```

See also

"F_FilePathOpenDir()" on page 551 and "F_FilePathCloseDir()" on page 548.

F_FilePathOpenDir()

`F_FilePathOpenDir()` allocates and returns a directory handle that you can use to call `F_FilePathGetNext()` to obtain the file and subdirectory entries in the directory.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
DirHandleT F_FilePathOpenDir(FilePathT *filepath,
    IntT *statusp);
```

Arguments

`filepath`	The filepath of the directory
`statusp`	A pointer to memory in which the function can store a status value

Returns

A handle to the directory, or `NULL` if the specified directory does not exist or is unreadable. `F_FilePathOpenDir()` returns `NULL` and sets the value of `statusp` to a nonzero value if there is not enough memory.

Example

See "F_FilePathGetNext()" on page 550 and "F_FilePathCloseDir()" on page 548.

F_FilePathParent()

`F_FilePathParent()` returns the filepath of the parent directory of a specified directory.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
FilePathT *F_FilePathParent(FilePathT *path,
    IntT *statusp);
```

Arguments

`path`	The filepath of the directory
`statusp`	A pointer to memory in which the function can store a status value

Returns

The filepath of the parent directory of the specified directory, or `NULL` if `path` specifies a root directory. If `F_FilePathParent()` fails, it returns `NULL` and sets `statusp` to a nonzero value.

Example

The following code prints the pathname of the `tmp` directory (the parent of the `mydir` directory):

```
. . .
IntT statusp;
FilePathT *path, *parentPath;
StringT pathname;
path = F_PathNameToFilePath("\<r\>\<c\>tmp\<c\>mydir",
                NULL, FDIPath);

parentPath = F_FilePathParent(path, &statusp);
pathname = F_FilePathToPathName(parentPath, FDIPath);
F_Printf(NULL, "%s\n", pathname);
. . .
```

F_FilePathProperty()

`F_FilePathProperty()` returns information about the file or directory specified by a filepath.

NOTE: This function works for HTTP paths as well. Property of the cached file is returned instead of the server file.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
IntT F_FilePathProperty(FilePathT *filepath,
   IntT pflags);
```

Arguments

`filepath`	The filepath of the file or directory.
`pflags`	Bit flags specifying the file or directory characteristics to evaluate. See the table below for a list of flags.

You can OR the following bit flags into `pflags`.

Bit mask	Information returned
`FF_FilePathReadable`	Whether `filepath` is readable
`FF_FilePathWritable`	Whether `filepath` is writable
`FF_FilePathDirectory`	Whether `filepath` is a directory
`FF_FilePathFile`	Whether `filepath` is a file
`FF_FilePathExist`	Whether `filepath` exists

Returns

A bit mask for the properties specified by `pflags`.

Example

The following code prints the string `The file is readable and writable` if the `tmp.txt` file in the `tmp` directory is readable and writable:

```
. . .
IntT pflags;
FilePathT *path;
pflags = FF_FilePathReadable | FF_FilePathWritable;
path = F_PathNameToFilePath("\<r\>\<c\>tmp\<c\>tmp.txt",
                NULL, FDIPath);
if(F_FilePathProperty(path, pflags) == pflags)
   F_Printf(NULL, "The file is readable and writable\n");
. . .
```

F_FilePathToPathName()

`F_FilePathToPathName()` converts a `FilePathT` filepath to a platform-dependent pathname string for a specified platform.

This function also supports HTTP paths.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
StringT F_FilePathToPathName(FilePathT *filepath,
    PathEnumT platform);
```

Arguments

`filepath`	The filepath to convert.
`platform`	The platform for which to produce a pathname. To avoid possible errors when you port your clients to different platforms, set `platform` to `FDefaultPath`.

Returns

The pathname, or `NULL` if `platform` specifies a platform other than the platform on which the client is currently running or if `F_FilePathToPathName()` fails to convert the pathname to a device-independent pathname.

`F_FilePathToPathName()` allocates the returned string with `F_Alloc()`. Use `F_Free()` to free this string when you are done with it.

Example

The following code creates a filepath from the device-independent pathname `\<r\>\<c\>tmp\<c\>tmp.txt`. It uses `F_FilePathToPathName()` to get a platform-specific pathname from the filepath.

```
. . .
FilePathT *path;
StringT pathname;
path = F_PathNameToFilePath("\<r\>\<c\>tmp\<c\>tmp.txt",
                NULL, FDIPath);
pathname = F_FilePathToPathName(path, FDefaultPath);
F_Printf(NULL, "%s\n", pathname);
F_Free(pathname);
. . .
```

Thiscode prints *current_drive_letter*`:\tmp\txt`.

.

F_FontEncId()

`F_FontEncId()` returns the ID for the named font encoding. Before calling this function, you must have called `F_FdeInitFontEncs()` to initialize the font encoding data.

This function supports UTF-8.

Synopsis

```
#include "fencode.h"
. . .
FontEncIdT F_FontEncId(ConStringT fontEncName);
```

Arguments

`fontEncName` The name of the font encoding to use.

Possible values for `fontEncName` are:

Value	Means
`FrameRoman`	Roman text
`JISX0208.ShiftJIS`	Japanese text
`BIG5`	Traditional Chinese text
`GB2312-80.EUC`	Simplified Chinese text
`KSC5601-1992`	Korean text

Returns

The ID of the font encoding you specified. If the FDE does not recognize `fontEncName` as a valid encoding name for the current session, this function returns the ID for the `FrameRoman` encoding.

Example

If the encoding for the current session's user onterface is `JISX0208.ShiftJIS`, the code returns the ID for that encoding. Otherwise, it returns the ID for the `FrameRoman` encoding.

```
FontEncIdT feId;
StringT encName;


F_FdeInit();
encName = F_ApiGetString(0, FV_SessionId,
FP_DialogEncodingName);
. . .
feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
```

The following code prints UTF-8

```
...
FontEncIdT feId;
F_FdeInitFontEncs((ConStringT) "UTF-8");
feId = F_TextEncToFontEnc(F_EncUTF8);
if (feId == F_FontEncId("UTF-8"))
F_Printf(NULL, F_FontEncName(feId));
...
```

F_FontEncName()

`F_FontEncName()` returns the name of the font encoding indicated by the specified `FontEncIdT`. Before calling this function, you must have called `F_FdeInitFontEncs()` to initialize the font encoding data.

This function can handle UTF-8 encoding.

Synopsis

```
#include "fencode.h"
. . .
ConStringT F_FontEncName(FontEncIdT fontEncId);
```

Arguments

`fontEncId` The ID of the font encoding to use.

Returns

The name of the font encoding associated with the specified `FontEncIdT`, or `NULL` if `fontEncId` is not a valid ID for the session.

Possible font encoding names are :

Value	Means
`FrameRoman`	Roman text
`JISX0208.ShiftJIS`	Japanese text
`BIG5`	Traditional Chinese text
`GB2312-80.EUC`	Simplified Chinese text
`KSC5601-1992`	Korean text
`UTF-8`	Unicode UTF-8 encoding

Example

```
. . .
FontEncIdT feId;
ConStringT encName

encName = F_FontEncName(feId);
. . .
/* Be sure to free encName when you are through */
```

F_FontEncToTextEnc()

Converts from `FontEncIdT` to `FTextEncodingT`

NOTE: This function has different behaviors for *Compatibility Mode* and *Unicode Mode*. In *Unicode Mode*, if no match is found, UTF-8 is the default output. In *Compatibility Mode*, the default output is FrameRoman.

Synopsis

```
#include "fencode.h"

. . .

FTextEncodingT F_FontEncToTextEnc (FontEncIdT feId);
```

Arguments

Returns

The text encoding corresponding to the font encoding

Example

The following code converts the string `myText` from the current Dialog Encoding to UTF-8 and prints it on the console:

```
#include "fencode.h"

. . .

F_FdeInit();

F_FdeInitFontEncs("UTF-8");/* Sets FDE to run in Unicode Mode */

F_ApiEnableUnicode(False); /* Sets APIs to run in Compatibility
Mode */

myText = F_ApiGetString(...); /* will return in Dialog Encoding
dlgEnc*/


encName = F_ApiGetString(0, FV_SessionId,
FP_DialogEncodingName);

FTextEncodintT dlgEnc =
F_FontEncToTextEnc(F_FontEncId(encName));

convertedText = F_StrConvertEnc(myText, dlgEnc, F_EncUTF8);


F_Printf(NULL, "%s", convertedText); /* convertedText is in UTF-
8 */

...
```

F_Free()

`F_Free()` frees the memory associated with a specified pointer.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
ErrorT F_Free(PtrT ptr);
```

Arguments

`ptr`	A pointer to the memory to free

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

See "F_Alloc()" on page 68.

F_FreeHandle()

`F_FreeHandle()` frees allocated memory for a handle that is not locked.

IMPORTANT: *Attempting to dereference a handle after freeing it may cause serious errors.*

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
ErrorT F_FreeHandle(HandleT handle);
```

Arguments

`handle`	The handle to free

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

See "F_AllocHandle()" on page 69.

F_GetFilePath()

`F_GetFilePath()` returns a platform-independent filepath associated with an open channel.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
FilePathT *F_GetFilePath(ChannelT channel);
```

Arguments

`channel`	The channel associated with the filepath

Returns

The platform-independent filepath associated with the specified channel, or `NULL` if it fails. Free the returned filepath with `F_FilePathFree()` when you are finished with it.

Example

The following code creates a temporary channel, gets the filepath associated with it, and prints its platform-specific pathname:

```
. . .
FilePathT *path;
StringT pathname;
ChannelT tmpChan;

/* Create the temporary channel. */
if((tmpChan = F_ChannelMakeTmp(1024)) == NULL)
   return;
path = F_GetFilePath(tmpChan); /* Get filepath from channel.*/
pathname = F_FilePathToPathName(path, FDefaultPath);
F_Printf(NULL, "The temporary channel is %s\n", pathname);
F_FilePathFree(path);
F_Free(pathname);
. . .
```

F_GetICUDataDir()

Gets the currently set ICU data directory for the calling process

NOTE: ICU data directory is set on a per-process basis. Certain types of clients, like synchronous DLL clients on Windows, reside in the same process space as FrameMaker. Such clients do not need to set the ICU data directory since FrameMaker sets the ICU data directory for its process. Thus, for such clients, F_GetICUDataDir will return the ICU data directory set for the FrameMaker process and therefore will return non-null even if the directory has not explicitly been set using F_SetICUDataDir in the client.

NOTE: The path returned by this call must not be freed.

Synopsis

```
#include "fencode.h"

. . .

ConStringT F_GetICUDataDir (VoidT);
```

Arguments

`VoidT`

Returns

The currently set ICU data directory path for the calling process

Example

The following code sets the ICU data directory of a client to `C:\icu_data` if it isn't set already:

```
...
ConStringT icu_dir = F_GetICUDataDir();
if (!icu_dir || !*icu_dir)
F_SetICUDataDir((ConStringT) "C:\\icu_data");
...
```

F_GetHandleSize()

`F_GetHandleSize()` returns the size of a handle's block of data.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
UIntT F_GetHandleSize(HandleT handle);
```

Arguments

`handle`	The handle to return the data block size of

Returns

The size of the handle's block of data in bytes.

Example

See "F_AllocHandle()" on page 69.

F_HandleEqual()

`F_HandleEqual()` determines whether blocks of memory specified by two handles have equal contents.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
BoolT F_HandleEqual(HandleT handle1,
   HandleT handle2);
```

Arguments

`handle1`	The first handle
`handle2`	The second handle

Returns

`True` if the blocks of memory specified by `handle1` and `handle2` have equal contents.

Example

The following code compares the contents of two blocks of memory:

```
. . .
HandleT hndl1, hndl2;

hndl1 = F_AllocHandle(66000, NO_DSE);
hndl2 = F_AllocHandle(66000, NO_DSE);

if(F_HandleEqual(hndl1, hndl2))
   F_Printf(NULL, "The handles are equal.\n");
. . .
```

F_HashCreate()

`F_HashCreate()` creates a hash table.

Synopsis

```
#include "fdetypes.h"
#include "fhash.h"
. . .
HashT F_HashCreate(StringT name, /* Name of the table */
   IntT minSize, /* Minimum size of the table */
   PShortT keyLen, /* Size of keys */
   GenericT notFound, /* Returned if searched key not found */
/* Determine if cell can be reused */
   BoolT (*deadQuery)(GenericT),
/* Called when cell is deleted */
   VoidT (*removeNotify)(GenericT),
/* Converts key to string*/
   Void(*T stringifyMe)(PtrT, UCharT *));
```

Arguments

`name`	The name of the table; useful for debugging.
`minSize`	A size suggestion for the memory allocation routines when the FDE creates the hash table. If you specify `0`, theFDE allocates memory according to its default calculations.
`keyLen`	The size, in bytes, of a hash key. For string keys, specify `KEY_IS_STRING`
`notFound`	The value for other `F_Hash` routines to return if they don't find a specific key.
`deadQuery`	A callback to determine if cell can be reused, or 0 (meaning, don't call any function here; assume that the table cell cannot be reused).
`removeNotify`	A callback to invoke when a value is being deleted,(e.g., during `F_HashRemove` and `F_HashDestroy`), or 0 (meaning no callback). You could use it, for example, to delete a data structure pointed to by the value.
`stringifyMe`	A function to produce a printable string from a key. Typically used for debugging.

IMPORTANT: *If the keys are strings, you must specify* `KEY_IS_STRING` *for* `keyLen` *and* `0` *for* `stringifyMe`*. If your keys are not strings, you must specify a value for* `keyLen`*, and you must specify a function for* `stringifyMe`*.*

The function specified by `deadQuery` is called as the hash routines maintain the hash table. You should only specify this function if your code can determine the validity of the cell's contents. To It is declared as:

```
BoolT deadQuery(GenericT datum);
```

The function specified by `removeNotify` is called when a cell is deleted. It provides an opportunity to free any data associated with the table cell. It is declared as:

```
VoidT removeNotify(GenericT datum);
```

Within this funciton, you can deallocate memory used by the values. (The FDK deallocates memory used by the keys.) If you don't need to deallocate memory for table cells, pass `0` for this argument.

If the key is not a string, you must specify a callback for `stringifyMe`. It is declared as:

```
VoidT stringifyMe(PtrT key, StringT info);
```

You can use the callback to convert the key to a string, or you can specify a function that always creates an empty string, for example:

```
VoidT noString(PtrT x, UCharT *y) {
        *y = 0;
}
```

Returns

The hash table, or `NULL` on failure.

Example

The following code creates a hash table and sets 26 entries. The data for each entry is a structure that contains a string and an integer. The code also shows a callback to free the data as each cell is deleted.

```
. . .
/* Structure for the cell data */
typedef struct {
   StringT type;
   IntT count;
} myDataT;

/* Callback to free cell data */
VoidT freeCell(GenericT datum)
{
   myDataT *data = (myDataT *) datum;
   F_Free(data->type);
   F_Free(data);
}
. . .

IntT i;
UCharT c;
UCharT s[2];
myDataT *data;
HashTableT *ht;

/* Create a hash table */
ht = F_HashCreate("myHash",0,KEY_IS_STRING,0,0,freeCell,0);
/* Populate the table with 26 cells. */
for(i=0;i<26;i++) {
   data = F_Alloc(sizeof(myDataT), DSE);
   F_ClearPtr(data, sizeof(myDataT));

   c = (UCharT)('A'+i);
   F_Sprintf(s, "%c", c);
   data->type = F_StrCopyString((StringT)"X");
   data->count = (IntT)0;
   F_HashSet(ht, s, (GenericT) data);
}
/* Get a cell from the table. */
```

```
data = (myDataT *)F_HashGet(ht, "A");
if(data)
   F_Printf(NULL, "\n%s: %d\n", data->type, data->count);
```

F_HashDestroy()

`F_HashDestroy()` deletes a hash table and all of its entries. If you specified a callback to handle cell deletion, the FDK calls that function for each cell `F_HashDestroy()` deletes.

Synopsis

```
#include "fdetypes.h"
#include "fhash.h"
. . .
VoidT F_HashDestroy(HashTableT *table);
```

Arguments

`table`	The hash table to delete

Returns

`VoidT`.

Example

The following code destroys a hash table:

```
. . .
HashTableT *mytable;
F_HashDestroy(mytable);
. . .
```

F_HashEnumerate()

`F_HashEnumerate()` accesses each table cell one by one, and calls a specified function with the key and datum of each cell; the order of access is arbitrary. The function you specify to handle the table cell should never delete the datum.

Synopsis

```
#include "fdetypes.h"
#include "fhash.h"
. . .
VoidT F_HashEnumerate(HashTableT *table,
    IntT (*proc)(PtrT, GenericT));
```

Arguments

`table`	The hash table
`proc`	The function to call

The function specified by `proc` should be defined as:

```
IntT proc(PtrT key,
    GenericT datum);
```

If this function returns `0`, enumeration stops.

Returns

`VoidT`.

Example

The following code prints out the data associated with each entry in the hash table:

```
. . .
IntT EnumFuncString(PtrT key, GenericT datum);
HashTableT *mytable;
. . .
F_HashEnumerate(mytable, (FunctionT)EnumFuncString);
. . .
IntT EnumFuncString(PtrT key, GenericT datum)
{
    myDataT *data = (myDataT *) datum;

    F_Printf(NULL, "Type: %s, count %d.\n",
                            data->type, data->count
    return True;
}
. . .
```

F_HashGet()

`F_HashGet()` fetches an entry from the hash table. The return value is of type `GenericT`, you must cast it to the desired type before you can use it.

Synopsis

```
#include "fdetypes.h"
#include "fhash.h"
. . .
GenericT F_HashGet(HashT table,
    PtrT key);
```

Arguments

`table`	The hash table
`key`	The key of the entry to get

Returns

The entry in the hash table with the specified key.

Example

For an example, see "F_HashCreate()" on page 564.

F_HashRemove()

`F_HashRemove()` removes an entry from the hash table. If you specified a callback to handle cell deletion, the FDK calls that function for each cell `F_HashRemove()` deletes.

Synopsis

```
#include "fdetypes.h"
#include "fhash.h"
. . .
VoidT F_HashRemove(F_HashT table,
    PtrT key);
```

Arguments

`table`	The hash table
`key`	The key of the entry to get

Returns

`VoidT`.

Example

The following code removes an entry from a hash table, then prints out the remaing contents of the table to the console:

```
. . .
VoidT EnumFuncString();
HashTableT *mytable;

F_HashRemove(mytable, (PtrT)"A");
F_HashEnumerate(mytable, EnumFuncString);

. . .
IntT EnumFuncString(PtrT key, GenericT datum)
{
    myDataT *data = (myDataT *) datum;

    F_Printf(NULL, "Type: %s, count %d.\n",
                            data->type, data->count
    return True;
}
. . .
```

F_HashReportOnData()

`F_HashReportOnData()` returns the number of hash entries with a specified datum. It prints the string information of the first matched datum to `info` by using the `stringifyMe` function specified by the call to `F_HashCreate()`. If more than one entry matching the specified datum is found, the string `" +"` is appended to the end of `info`.

Synopsis

```
#include "fdetypes.h"
#include "fhash.h"
. . .
IntT F_HashReportOnData(HashT table,
    StringT info,
    GenericT datum);
```

Arguments

`table`	The hash table
`info`	A pointer to memory that the `stringifyMe` function can write to
`datum`	The datum to search for

Returns

The number of entries in the hash table with the specified datum.

Example

The following code reports the number of entries with datum of `1`:

```
. . .
UCharT info[256];
IntT num_entries;
HashTableT *ht =
          F_HashCreate("color", 0, KEY_IS_STRING, 0, 0, 0, 0);

F_HashSet(ht, (StringT) "red", 1);
F_HashSet(ht, (StringT) "green", 2);
F_HashSet(ht, (StringT) "blue", 3);
F_HashSet(ht, (StringT) "rojo", 1);
F_HashSet(ht, (StringT) "rouge", 1);

num_entries = F_HashReportOnData(ht, info, 1);
F_Printf(0, "Number of entries found with datum %d: %d\n",
1, num_entries);

F_HashDestroy(ht);
. . .
```

F_HashSet()

Adds the specified data to the specified hash table, giving it the specified key. This function makes its own copy of the key, but it does not make a copy of the data.

IMPORTANT: *The routine that copies the key only copies strings or the amount of bytes specified for keyLen in F_HashCreate(). If you use pointers in your keys, F_HashSet() will only copy the pointer, not the pointer data.*

Synopsis

```
#include "fdetypes.h"
#include "fhash.h"
. . .
ErrorT F_HashSet(HashT table,
    PtrT key,
    GenericT datum);
```

Arguments

`table`	The hash table
`key`	The key of the entry to set
`datum`	The entry's datum

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

See "F_HashCreate()" on page 564.

F_IsValidUTF8()

Determines whether the specified byte sequence is a valid UTF-8 string

Synopsis

```
#include "fencode.h"

. . .

BoolT F_IsValidUTF8 (ConStringT s);
```

Arguments

`s`	The string to check

Returns

A nonzero value if the byte sequence is a valid UTF-8 string, or 0 if it isn't

Example

The following code prints `First is valid, Second is invalid`

```
#include "fencode.h"
. . .
StringT first = "\x41\xE2\x80\x93\x42";
StringT second = "\x41\xE2\x80";
F_FdeInitFontEncs((ConStringT)"UTF-8");
if (F_IsValidUTF8(first))
   F_Printf(NULL,"First is valid");
if (!F_IsValidUTF8(first))
   F_Printf(NULL,"Second is invalid");
...
```

F_LanguageNumber()

`F_LanguageNumber()` returns the Frame API number constant that corresponds to a language string, such as `usenglish` or `deutsch`.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_LanguageNumber(ConStringT langName);
```

Arguments

`langName`	The language string for which to get a constant

Returns

The number constant that corresponds to the specified language string.

The following table lists the API language strings and the corresponding constants returned by `F_LanguageNumber()`.

Language string	API language constant
usenglish	FV_LANG_ENGLISH
ukenglish	FV_LANG_BRITISH
deutsch	FV_LANG_GERMAN
swissgerman	FV_LANG_SWISS_GERMAN
francais	FV_LANG_FRENCH
canadianfrench	FV_LANG_CANADIAN_FRENCH
espanol	FV_LANG_SPANISH
catalan	FV_LANG_CATALAN
italiano	FV_LANG_ITALIAN
portuguese	FV_LANG_PORTUGUESE
brasil	FV_LANG_BRAZILIAN
danish	FV_LANG_DANISH
dutch	FV_LANG_DUTCH
norwegian	FV_LANG_NORWEGIAN
nynorsk	FV_LANG_NYNORSK
finnish	FV_LANG_FINNISH
svenska	FV_LANG_SWEDISH
japanese	FV_LANG_JAPANESE
traditionalchinese	FV_LANG_TRADITIONAL_CHINESE
simplifiedchinese	FV_LANG_SIMPLIFIED_CHINESE
korean	FV_LANG_KOREAN

Example

The following code prints `The language is U.K. English`.

```
. . .
if(FV_LANG_BRITISH == F_LanguageNumber("ukenglish"))
   F_Printf(NULL, "The language is U.K. English.");
. . .
```

F_LanguageString()

`F_LanguageString()` returns the language string specified by a Frame API constant, such as `FV_LANG_ENGLISH` or `FV_LANG_GERMAN`. For a list of the language strings and the corresponding constants, see "F_LanguageNumber()" on page 574.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
ConStringT F_LanguageString(IntT langConstant);
```

Arguments

`langConstant` The constant for which to get a language string

Returns

The language string specified by the constant. `F_LanguageString()` returns `NULL` if `langConstant` is not one of the API constants that specify a language. Do not free the returned string when you are done with it.

Example

The following code prints the string `usenglish`:

```
. . .
ConStringT langString;

langString = F_LanguageString(FV_LANG_ENGLISH);
F_Printf(NULL, "%s\n", langString);
. . .
```

F_LockHandle()

`F_LockHandle()` locks the memory associated with a specified handle so that it can't be relocated. `F_LockHandle()` returns the memory address of the handle's block of memory so that you can access it.

`AddrT` is a super type of `PtrT`. You should not cast it to `PtrT`. However, you can cast `PtrT` to `AddrT`. The `AddrT` type is provided primarily for very large memory blocks on Windows platforms. Accessing `AddrT` is slower than accessing `PtrT` on Windows platforms.

After you have locked a handle, you can't use `F_FreeHandle()` or `F_ReallocHandle()` on the handle.

`F_LockHandle()` increases the lock count.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
AddrT F_LockHandle(HandleT handle);
```

Arguments

`handle`	The handle to lock

Returns

The memory address of the data block of the handle if it succeeds, or `NULL` if fails.

Example

The following code locks a handle:

```
. . .
HandleT hndl = NULL;

if (F_LockHandle(hndl) == NULL)
    F_Printf(NULL, "Couldn't lock handle.\n");
. . .
```

See also

"F_AllocHandle()" on page 69 and "F_UnlockHandle()" on page 714.

F_MakeDir()

`F_MakeDir()` creates a directory specified by a filepath. If the permission to create a directory is denied, `F_MakeDir()` fails. If the filepath specifies an existing file or directory, its parent directory is not writable, or if its ancestor directories do not exist, `F_MakeDir()` fails.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. .
ErrorT F_MakeDir(FilePathT *filepath);
```

Arguments

`filepath`	The filepath of the directory to create

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

The following code creates a directory named `adir` in the `tmp` directory:

```
. . .
FilePathT *path;

path = F_PathNameToFilePath((StringT)"<r><c>tmp<c>adir",
                    NULL, FDIPath);
if(F_MakeDir(path)!= FdeSuccess)
{
    F_Printf(NULL, "Couldn't create directory.\n");
    return;
}
. . .
```

F_MetricApproxEqual()

`F_MetricApproxEqual()` compares two metric numbers.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
BoolT F_MetricApproxEqual(MetricT x,
   MetricT y);
```

Arguments

`x`	A metric number to compare with `y`
`y`	A metric number to compare with `x`

Returns

`True` if the difference of `x` and `y` is less than the predefined small value `FM_EPSILON` in the FrameMaker product.

Example

The following code gets the width of the first selected object in the current document. If its width is approximately equal to one inch, it prints the string `The width is approximately one inch`.

```
. . .
#define IN ((MetricT) 65536*72)

F_ObjHandleT docId, objId;
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
objId = F_ApiGetId(FV_SessionId, docId,
FP_FirstSelectedGraphicInDoc);
if(F_MetricApproxEqual(F_ApiGetMetric(docId, objId, FP_Width),
                       IN))
   F_Printf(NULL, "The width is approximately one inch\n");
. . .
```

F_MetricConstrainAngle()

`F_MetricConstrainAngle()` constrains a specified angle to a specified range of degrees.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
VoidT F_MetricConstrainAngle(AngleT *anglep,
    AngleT lbound);
```

Arguments

`anglep`	The address of the angle to constrain.
`lbound`	The lower bound of the range of degrees to which to constrain the angle. `F_MetricConstrainAngle()` constrains the angle specified by `anglep` between `lbound` and `lbound+360` degrees.

The `AngleT` type is the same as `MetricT`.

Returns

`VoidT`.

Example

The following code constrains an angle of 800 degrees. It prints the string `440.00`.

```
. . .
#define DEG (AngleT) 65536

AngleT angle = 800*DEG;
F_MetricConstrainAngle(&angle, 180*DEG);
F_Printf(NULL, "%2.2f\n", F_MetricToFloat((MetricT) angle));
. . .
```

F_MetricDiv()

`F_MetricDiv()` divides metric numbers.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
MetricT F_MetricDiv(MetricT x,
   MetricT y);
```

Arguments

`x`	The dividend
`y`	The divisor

Returns

The quotient of the specified numbers.

Example

The following code prints the string `0x20000`:

```
. . .
#define PTS (MetricT) 655362F_Printf(NULL, "0x%x\n",
F_MetricDiv(4*PTS, 2*PTS));
. . .
```

F_MetricFloat()

`F_MetricFloat()` converts a real number to a metric number.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
MetricT F_MetricFloat(PRealT f);
```

Arguments

`f`	The real number to convert

Returns

The metric number.

Example

The following code prints the string `0x10006`:

```
. . .
F_Printf(NULL, "0x%x\n", F_MetricFloat((PRealT)1.0001));
. . .
```

F_MetricFractMul()

`F_MetricFractMul()` multiplies the metric number `x` by `n/d`, where `n` is a nonnegative and `d` is a positive integer. `F_MetricFractMul()` provides greater accuracy than `F_MetricMul()` when you are multiplying fractional numbers.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
MetricT F_MetricFractMul(MetricT x,
    IntT n,
    IntT d);
```

Arguments

`x`	The metric number to multiply
`n`	The numerator of the fraction to multiply by `x`
`d`	The denominator of the fraction to multiply by `x`

Returns

The product of `x` multiplied by `n/d`.

Example

The following code turns on the view grid of the active document and sets the grid units to .25 inches:

```
. . .
#define IN ((MetricT) 65536*72)

F_ObjHandleT docId;
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
F_ApiSetInt(FV_SessionId, docId, FP_ViewGrid, True);
F_ApiSetMetric(FV_SessionId, docId, FP_ViewGridUnits,
          F_MetricFractMul(IN, 1, 4));
. . .
```

F_MetricMake()

`F_MetricMake()` constructs a metric number from a fraction.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
MetricT F_MetricMake(IntT numerator,
    IntT denominator);
```

Arguments

`numerator`	The fraction's numerator.
`denominator`	The fraction's denominator. If it is 0, the FDE assert fails.

Returns

The metric number constructed from the two specified numbers.

Example

The following code sets the zoom of the current document to 125%:

```
. . .
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
F_ApiSetMetric(FV_SessionId, docId, FP_Zoom,
          F_MetricMake(5, 4));
. . .
```

F_MetricMul()

`F_MetricMul()` multiplies the metric numbers `x` and `y`.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
MetricT F_MetricMul(MetricT x,
   MetricT y);
```

Arguments

`x`	The first metric number
`y`	The second metric number

Returns

The result of the two multiplied numbers.

Example

The following code prints the string `0x60000`:

```
. . .
#define PTS (MetricT) 65536

F_Printf(NULL, "0x%x\n", F_MetricMul(3*PTS, 2*PTS));
. . .
```

F_MetricNormalizeAngle()

`F_MetricNormalizeAngle()` normalizes a specified angle between 0 and 360 degrees.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
VoidT F_MetricNormalizeAngle(AngleT *anglep);
```

Arguments

`anglep`	The address of the angle to normalize

Returns

`VoidT`.

Example

The following code normalizes an angle of 800 degrees. It prints the string `80.00`.

```
. . .
#define DEG (AngleT) 65536
AngleT angle = 800*DEG;

F_MetricNormalizeAngle(&angle);
F_Printf(NULL, "%2.2f\n", F_MetricToFloat((MetricT) angle));
. . .
```

F_MetricSqrt()

`F_MetricSqrt()` computes the square root of a metric number.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
MetricT F_MetricSqrt(MetricT x);
```

Arguments

`x`	The metric number for which to compute the square root

Returns

The square root of the specified number.

Example

The following code prints the string `0x20000`:

```
. . .
#define PTS (MetricT) 65536

F_Printf(NULL, "0x%x\n", F_MetricSqrt(4*PTS));
. . .
```

F_MetricSquare()

`F_MetricSquare()` computes the square of a metric number.

`F_MetricSquare()` returns a negative number if an overflow error occurs.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
MetricT F_MetricSquare(MetricT x);
```

Arguments

`x`	The metric number to square

Returns

The square of the specified number.

Example

The following code prints the string `0x40000`:

```
. . .
#define PTS (MetricT) 65536

F_Printf(NULL, "0x%x\n", F_MetricSquare(2*PTS));
. . .
```

F_MetricToFloat()

`F_MetricToFloat()` converts a metric number to a real number.

Synopsis

```
#include "fdetypes.h"
#include "fmetrics.h"
. . .
PRealT F_MetricToFloat(MetricT m);
```

Arguments

`m`	The metric number to convert

Returns

The real number corresponding to the specified metric number.

Example

The following code prints the string `3.0000`:

```
. . .
#define PTS (MetricT) 65536

F_Printf(NULL, "%4.4f\n", F_MetricToFloat(3*PTS));
. . .
```

F_MifBegin()

`F_MifBegin()` indents and starts a new MIF statement, and then automatically increases the indent level.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifBegin(StringT text);
```

Arguments

`text`	The MIF statement to write to the MIF channel

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifSetIndent(0);
F_MifBegin("ColorCatalog");
    F_MifBegin("Color");
        F_MifBegin("ColorTag");
            F_MifTextString((StringT)"Black");
        F_MifEnd("ColorTag");
    F_MifEnd("Color");
F_MifEnd("ColorCatalog");
. . .
```

generates the following output to the MIF channel:

```
<ColorCatalog
    <Color
        <ColorTag `Black' >
    > # end of Color
> # end of ColorCatalog
```

F_MifComment()

`F_MifComment()` writes a comment string to the MIF write channel.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifComment(StringT s);
```

Arguments

`s`	The comment string to write to the MIF channel

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifSetIndent(0);
F_MifMIFFile(5.0);
F_MifComment((StringT) "Generated by FDK Client");
. . .
```

generates the following output to the MIF channel:

```
<MIFFile 5.0> # Generated by FDK Client
```

F_MifDecimal()

`F_MifDecimal()` writes a real number with `n` digits after the decimal point.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifDecimal(PRealT d,
    IntT n,
    MifUnitT unit);
```

Arguments

`s`	The real number to write to the MIF channel.
`n`	The number of digits after the decimal point to be printed.
`unit`	The MIF unit. See the table below.

The type `MifUnitT` can have the following values.

MifUnitT value	Measurement unit
`MIFUnitIn`	Inches
`MIFUnitCm`	Centimeters
`MIFUnitMm`	Millimeters
`MIFUnitPica`	Picas
`MIFUnitPt`	Points
`MIFUnitDd`	Didots
`MIFUnitCc`	Ciceros
`MIFUnitDef`	Default unit

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifDecimal(3.1415, 3, MIFUnitCm);
. . .
```

generates the following output to the MIF channel:

```
3.142 cm
```

F_MifEnd()

`F_MifEnd()` indents and finishes a MIF statement, and then automatically decreases the indent level. If the MIF statement has substatements, the function outputs the comment string `# end of` *`statement`*, where *`statement`* is the name of the substatement.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifEnd(StringT text);
```

Arguments

`text`	The text to write to the MIF channel

Returns

`VoidT`.

Example

See “F_MifBegin()” on page 589.

F_MifGetIndent()

`F_MifGetIndent()` returns the current indent level of the MIF write channel.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
IntT F_MifGetIndent(VoidT);
```

Arguments

None.

Returns

`VoidT`.

Example

The following code writes the string `Indent is 2 levels`:

```
. . .
F_MifSetIndent(2);
F_Printf(NULL, "Indent is %d levels\n", F_MifGetIndent());
. . .
```

F_MifIndent()

`F_MifIndent()` indents the output channel according to the indent level.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifIndent(VoidT);
```

Arguments

None.

Returns

`VoidT`.

Example

The following code indents the output channel according to the current indent level:

```
. . .
F_MifIndent(VoidT);
. . .
```

F_MifIndentDec()

`F_MifIndentDec()` decreases the indent level.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
IntT F_MifIndentDec(VoidT);
```

Arguments

None.

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifBegin("Pgf");
F_MifBegin("PgfFont");
F_MifIndentDec();
F_MifBegin("FSize");
F_MifDecimal(12.0, 1, MIFUnitPt);
F_MifEnd("FSize");
F_MifEnd("PgfFont");
F_MifEnd("Pgf");
. . .
```

generates the following output to the MIF channel:

```
<Pgf
    <PgfFont
    <FSize 12.0 pt >
> # end of PgfFont
> # end of Pgf
```

F_MifIndentInc()

`F_MifIndentInc()` increases the indent level.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
IntT F_MifIndentInc(VoidT);
```

Arguments

None.

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifBegin("Pgf");
F_MifIndentInc();
F_MifBegin("PgfSpBefore");
F_MifDecimal(0.0, 1, MIFUnitPt);
F_MifEnd("PgfSpBefore");
F_MifEnd("Pgf");
. . .
```

generates the following output to the MIF channel:

```
<Pgf
       <PgfSpBefore 0.0 pt >
   > # end of Pgf
```

F_MifInteger()

`F_MifInteger()` writes an integer to the MIF write channel.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifInteger(IntT n);
```

Arguments

`n`	The integer to write to the MIF channel

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifInteger(24);
. . .
```

generates the following output to the MIF channel:

```
24
```

F_MifNewLine()

`F_MifNewLine()` writes a new line to the MIF write channel.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifNewLine(VoidT);
```

Arguments

None.

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifSetIndent(0);
F_MifMIFFile(5.0);
F_MifNewLine();
F_MifComment((StringT)"Options");
F_MifNewLine();
F_MifComment((StringT)"   Paragraph Tags");
. . .
```

generates the following output to the MIF channel:

```
<MIFFile 5.0 >
# Options
#    Paragraph Tags
```

F_MifSetIndent()

`F_MifSetIndent()` sets the indent level of the MIF write channel.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifSetIndent(IntT indent);
```

Arguments

`indent`	The indent level to set

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifMIFFile(5.0);
F_MifSetIndent(3);
F_MifBegin("ColorCatalog");
F_MifEnd("ColorCatalog");
. . .
```

generates the following output to the MIF channel:

```
<MIFFile 5.00 >
          <ColorCatalog  >
```

F_MifSetOutputChannel()

`F_MifSetOutputChannel()` sets a channel to receive MIF output.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
ChannelT F_MifSetOutputChannel(ChannelT chan);
```

Arguments

`chan`	The channel to set as the MIF output channel

Returns

The channel previously set as the MIF output channel.

Example

The following code:

```
. . .
FilePathT *path;
ChannelT chan;

path = F_PathNameToFilePath((StringT)"my.mif",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"w")) == NULL) return;
F_MifSetOutputChannel(chan);
F_MifSetIndent(0);
F_MifMIFFile(5.0);
F_ChannelClose(chan);
. . .
```

sets the MIF output channel to the file `my.mif` in the FrameMaker product directory. It generates the following output to the file:

```
<MIFFile 5.00 >
```

F_MifSpace()

`F_MifSpace()` writes a blank space to the MIF output channel.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifSpace(VoidT);
```

Arguments

None.

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifText((StringT)"Some text");
F_MifSpace();
F_MifText((StringT)"More text");
. . .
```

generates the following output to the MIF channel:

```
Some text More text
```

F_MifTab()

`F_MifTab()` writes a tab to the MIF channel.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifTab(VoidT);
```

Arguments

None.

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifText((StringT)"Some text");
F_MifTab();
F_MifText((StringT)"More text");
. . .
```

generates the following output to the MIF channel:

```
Some Text     More text
```

F_MifText()

`F_MifText()` writes a simple text string to the MIF output channel.

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifText(StringT s);
```

Arguments

`s`	The text string to write

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifText((StringT)"Some text");
. . .
```

generates the following output to the MIF channel:

```
Some text
```

F_MifTextString()

`F_MifTextString()` writes a text string enclosed in single quotes (`'`) to the MIF channel.

IMPORTANT: This function only works with characters that use the FrameRoman encoding. This is important if your document includes Asian text. For a function to write double-byte characters out to MIF, see "F_MifText()" on page 600

Synopsis

```
#include "fdetypes.h"
#include "fmifstmt.h"
. . .
VoidT F_MifTextString(StringT s);
```

Arguments

`s`	The text string to write

Returns

`VoidT`.

Example

The following code:

```
. . .
F_MifTextString((StringT)"This is a string");
. . .
```

generates the following output to the MIF channel:

```
`This is a string'
```

Simple MIF library

Simple MIF library functions are useful for writing MIF statements.

The simple MIF library functions write directly to the MIF channel. To use them, you must first open a channel with `F_ChannelOpen()` and then set it as the MIF channel

with `F_MifSetOutputChannel()`. For more information on using MIF channels, see "The MIF library" on page 553 of the *FDK Programmer's Guide*.

```
#include "fmifstmt.h"
. . .

VoidT F_MifMIFFile(PRealT version)

VoidT F_MifString(StringT string);

VoidT F_MifDMargins(PRealT l,
       PRealT t,
       PRealT r,
       PRealT b,
       MifUnitT unit);

VoidT F_MifDColumns(IntT DColumns);

VoidT F_MifDColumnGap(PRealT DColumnGap,
       MifUnitT unit);

VoidT F_MifDPageSize(PRealT w,
       PRealT h,
       MifUnitT unit);

VoidT F_MifDStartPage(IntT DStartPage);

VoidT F_MifDTwoSides(BoolT DTwoSides);

VoidT F_MifDParity(IntT DParity);

VoidT F_MifDPageNumStyle(IntT DPageNumStyle);

VoidT F_MifDFNotePgfTag(StringT DFNotePgfTag);

VoidT F_MifDFNoteMaxH(PRealT DFNoteMaxH,
       MifUnitT unit);

VoidT F_MifDFNoteNumStyle(IntT DFNoteNumStyle);

VoidT F_MifDFNoteLabels(StringT DFNoteLabels);

VoidT F_MifDFNotesNumbering(IntT DFNotesNumbering);

VoidT F_MifDFNoteNumberPos(IntT DFNoteNumberPos);

VoidT F_MifDPageRounding(IntT DPageRounding);

VoidT F_MifDLinebreakChars(StringT DLinebreakChars);

VoidT F_MifPgfTag(StringT tagStyleName);

VoidT F_MifPgfUseNextTag(BoolT pgfUseNextTag);

VoidT F_MifPgfNextTag(StringT pgfNextTag);
```

```
VoidT F_MifPgfLIndent(PRealT pgfLIndent,
       MifUnitT unit);

VoidT F_MifPgfFIndent(PRealT pgfFIndent,
       MifUnitT unit);

VoidT F_MifPgfRIndent(PRealT pgfRIndent,
       MifUnitT unit);

VoidT F_MifPgfAlignment(IntT pgfAlignment);

VoidT F_MifPgfTopSeparator(StringT pgfTopSeparator);

VoidT F_MifPgfBotSeparator(StringT pgfBotSeparator);

VoidT F_MifPgfPlacement(IntT pgfPlacement);

VoidT F_MifPgfSpBefore(PRealT pgfSpBefore,
       MifUnitT unit);

VoidT F_MifPgfSpAfter(PRealT pgfSpAfter,
       MifUnitT unit);

VoidT F_MifPgfWithNext(BoolT pgfWithNext);

VoidT F_MifPgfBlockSize(IntT pgfBlockSize);

VoidT F_MifPgfAutoNum(BoolT pgfAutoNum);

VoidT F_MifPgfNumberFont(StringT fTag);

VoidT F_MifPgfNumFormat(StringT pgfNumFormat);

VoidT F_MifPgfNumString(StringT pgfNumString);

VoidT F_MifPgfLineSpacing(IntT pgfLineSpacing);

VoidT F_MifPgfLeading(PRealT pgfLeading,
       MifUnitT unit);

VoidT F_MifPgfNumTabs(IntT pgfNumTabs);

VoidT F_MifTSX(PRealT tsx,
       MifUnitT unit);

VoidT F_MifTSType(IntT tsType);

VoidT F_MifTSLeaderStr(StringT tsLeader);

VoidT F_MifTSDecimalChar(IntT decimalChar);

VoidT F_MifPgfHyphenate(BoolT pgfHyphenate);

VoidT F_MifHyphenMaxLines(IntT hyphenMaxLines);

VoidT F_MifHyphenMinPrefix(IntT hyphenMinPrefix);
```

```
VoidT F_MifHyphenMinSuffix(IntT hyphenMinSuffix);

VoidT F_MifHyphenMinWord(IntT hyphenMinWord);

VoidT F_MifHyphenQuality(StringT hyphenQuality);

VoidT F_MifPgfLetterSpace(BoolT pgfLetterSpace);

VoidT F_MifPgfMinWordSpace(IntT pgfMinWordSpace);

VoidT F_MifPgfOptWordSpace(IntT pgfOptWordSpace);

VoidT F_MifPgfMaxWordSpace(IntT pgfMaxWordSpace);

VoidT F_MifPgfLanguage(IntT pgfLanguage);

VoidT F_MifPgfCellAlignment(IntT valign);

VoidT F_MifPgfCellMargins(PRealT b,
       PRealT t,
       PRealT r,
       PRealT b,
       MifUnitT unit);

VoidT F_MifFTag(StringT fTag);

VoidT F_MifFFamily(StringT fFamily);

VoidT F_MifFVar(StringT fVar);

VoidT F_MifFWeight(StringT fWeight);

VoidT F_MifFAngle(StringT fAngle);

VoidT F_MifFSize(PRealT fSize,
       MifUnitT unit);

VoidT F_MifFUnderline(BoolT fUnderline);

VoidT F_MifFStrike(BoolT fstrike);

VoidT F_MifFSupScript(BoolT fSupScript);

VoidT F_MifFSubScript(BoolT fSubScript);

VoidT F_MifFSmallCaps(BoolT fsmallcaps);

VoidT F_MifFCaps(BoolT fcaps);

VoidT F_MifFChangeBar(BoolT fChangeBar);

VoidT F_MifFOutline(BoolT foutline);

VoidT F_MifFShadow(BoolT fshadow);

VoidT F_MifFPairKern(BoolT fPairKern);
```

```
VoidT F_MifFPlain(BoolT fPlain);

VoidT F_MifFBold(BoolT fbold);

VoidT F_MifFItalic(BoolT fitalic);

VoidT F_MifFDX(PRealT fDX,
       MifUnitT unit);

VoidT F_MifFDY(PRealT fDY,
       MifUnitT unit);

VoidT F_MifFDW(PRealT fDW,
       MifUnitT unit);

VoidT F_MifFLocked(BoolT fLocked);

VoidT F_MifFSeparation(IntT fSeparation);

VoidT F_MifFlipLR(BoolT flipLR);

VoidT F_MifPen(IntT pen);

VoidT F_MifFill(IntT fill);

VoidT F_MifPenWidth(PRealT penWidth);

VoidT F_MifSeparation(IntT layer);

VoidT F_MifAngle(IntT angle);

VoidT F_MifBRect(PRealT l,
       PRealT t,
       PRealT w,
       PRealT h,
       MifUnitT unit);

VoidT F_MifBeginTextFlow(StringT flowTag,
       BoolT autoConnect);

VoidT F_MifEndTextFlow(VoidT);

VoidT F_MifTextRectID(IntT textRectID);

VoidT F_MifID(IntT id);

VoidT F_MifPageSize(PRealT w,
       PRealT h,
       MifUnitT unit);

VoidT F_MifPageType(IntT pageType);

VoidT F_MifPageTag(StringT pageTag);

VoidT F_MifPageOrientation(IntT pageOrientation);
```

```
VoidT F_MifPageNum(IntT pageNum);

VoidT F_MifPageBackground(StringT pageTag);

VoidT F_MifTag(StringT tag);

VoidT F_MifFrameType(IntT frameType);

VoidT F_MifHLine(IntT pen,
       PRealT penWidth,
       PRealT x,
       PRealT y,
       PRealT length);

VoidT F_MifNumPoints(IntT n);

VoidT F_MifPoint(PRealT x,
       PRealT y,
       MifUnitT unit);

VoidT F_MifTLOrigin(PRealT x,
       PRealT y,
       MifUnitT unit);

VoidT F_MifTLAlignment(IntT align);

VoidT F_MifChar(IntT text);

VoidT F_MifVariable(StringT text);

VoidT F_MifUnits(IntT u);
```

F_PathNameToFilePath()

`F_PathNameToFilePath()` constructs a platform-independent `FilePathT` path from a platform-specific pathname.

This function also supports HTTP paths.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
FilePathT *F_PathNameToFilePath(StringT pathname,
   FilePathT *anchor,
   PathEnumT platform);
```

Arguments

`pathname`	The pathname to convert to a filepath.
`anchor`	An anchor for the filepath. If `pathname` specifies a relative pathname, `F_PathNameToFilePath()` constructs the path relative to the anchor specified by `anchor`. If `pathname` specifies an absolute pathname, `F_PathNameToFilePath()` ignores the anchor specified by `anchor`.
`platform`	The platform for the pathname. To make your client portable and avoid errors, specify `FDefaultPath` or `FDIPath`.

If `anchor` is `NULL`, `F_PathNameToFilePath()` constructs the path relative to the current directory.

Returns

The platform-independent `FilePathT` path based on the path specified by `pathname`, or `NULL` if the path specified by `pathname` is invalid or inconsistent with the specified platform.

Example

The following code creates filepaths from pathnames:

```
. . .
FilePathT *anchor, *path1, *path2, *path3, *path4;

path1 = F_PathNameToFilePath("<c>my.txt",
                    NULL, FDIPath);
path2 = F_PathNameToFilePath("my.txt",
                    NULL, FDefaultPath);

anchor = F_PathNameToFilePath("<r><c>tmp",
                    NULL, FDIPath);
path3 = F_PathNameToFilePath("<r><c>tmp<c>tmp.txt",
                    NULL, FDIPath);
path4 = F_PathNameToFilePath("tmp.txt",
                    anchor, FDefaultPath);
. . .
F_FilePathFree(path1);
F_FilePathFree(path2);
F_FilePathFree(path3);
F_FilePathFree(path4);
F_FilePathFree(anchor);
. . .
```

`path1` and `path2` specify a file named `my.txt` in the FrameMaker product directory. `path3` and `path4` specify a file named `tmp.txt` in the `tmp` directory.

F_PathNameType()

`F_PathNameType()` uses a set of heuristics to guess the path type of the path specified by `pathname`. It returns the most likely path type. This path type may not always be the correct one because of ambiguities in pathnames among the different platforms.

NOTE: This function returns FDosPath type for HTTP paths. The API `F_PathNameType()` logically translates to the API `F_PathNameValid()` that returns `FDosPath`, if the path is valid.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
PathEnumT F_PathNameType(StringT pathname);
```

Arguments

`pathname`	The pathname for which to guess the path type

Returns

A constant specifying the pathname type (`FDosPath`).

Example

The following code returns the pathname types for various pathnames:

```
. . .
PathEnumT pType; /* FMacPath, FDosPath, FUnixPath, or FDIPath */

pType = F_PathNameType("HD:folder"); /* Returns FMacPath. */
pType = F_PathNameType("c:\\bin"); /* Returns FDosPath. */
pType = F_PathNameType("/tmp"); /* Returns FUnixPath. */
pType = F_PathNameType("<c>file"); /* Returns FDIPath. */
. . .
```

F_Printf()

`F_Printf()` prints formatted output to the Frame console .

`F_Printf()` is similar to `fprintf()`, except it doesn't support the `p` conversion character. All the arguments that you pass to `F_Printf()` must be FDE types and should be typecast.

This function can accept the %C escape sequence. In Compatibility Mode, this ignores the corresponding parameter. In Unicode Mode, the first UTF-8 character in the corresponding parameter (which must be ConStringT or UCharT *) is printed.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
IntT F_Printf(ChannelT channel,
   NativeCharT *format,
   ... );
```

Arguments

`channel`	The channel to which to print the output. To print to the the Frame console specify `NULL`.
`format`	The formatted string to print.

You can precede the conversion character (`d`, `o`, `x`, or `u`) with `l` or `h`. To specify `IntT`, precede the conversion character with `l`. To specify `ShortT`, precede it with `h`. By default, the conversion characters `d`, `o`, `x`, and `u` specify `IntT`.

IMPORTANT: If you don't typecast the arguments you pass to `F_Printf()`, the output it prints may be different on different platforms.

Returns

The number of matched characters printed.

Examples

The following code prints `100 200 strings`:

```
. . .
F_Printf(NULL, "%d %hd %s\n", (IntT) 100, (ShortT)200,
          (StringT) "strings");
. . .
```

The following code prints ४ + २ = 6

```
...
StringT devanagiri_four="\xE0\xA5\xAA";
StringT devanagiri_two ="\xE0\xA5\xA8";
IntT res;
F_FdeInitFontEncs((ConStringT)"UTF-8");
res = F_DigitValue(devanagiri_four)
+F_DigitValue(devanagiri_two);
F_Printf(NULL,"%C + %C is %d", devanagiri_four, devanagiri_two,
res);
...
```

F_Progress()

`F_Progress()` is a function you can use to indicate what percentage of a process has been completed. It displays a progress indicator with a "thermometer" showing the percentage done.

Synopsis

```
#include "fprogs.h"
. .
ErrorT F_Progress(IntT percent);
```

Arguments

`percent`	An integer representing a percentage; for example, 30 represents 30%.

Returns

`FeSuccess` if it succeeds, or a nonzero value if it fails.

Example

The following code responds to a file-to-file fliter notification, and displays a progress indicator. If the user does not cancel, then the code will go on to perform the filter operation.

```
. . .
VoidT
F_ApiNotify(IntT notification, F_ObjHandleT docId,
          StringT filename, IntT iparm)
{
```

```
if (notification == FA_Note_FilterFileToFile) {
    F_FilterArgsT *argsp = (F_FilterArgsT *) filename;

    IntT error;

    F_ApiSleep(2);  /* give user time to react */
    error = F_Progress(10);
    if (error != 0)   {
        /* return the error result so FrameMaker can */
        /* cancel the operation. */
        F_ApiReturnValue(-1);
    /* Go on to perform the filter operation, calling */
    /* F_Progress() at appropriate intervals. */
. . .
```

F_PtrEqual()

`F_PtrEqual()` determines whether two blocks of memory have equal contents to a specified number of bytes.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
BoolT F_PtrEqual(PtrT ptr1,
    PtrT ptr2,
    UIntT n);
```

Arguments

`ptr1`	A pointer to a block of memory to compare
`ptr2`	A pointer to a block of memory to compare
`n`	The number of bytes of memory to compare

Returns

`True` if the blocks of memory have equal contents to `n` bytes, or `False` if they do not have equal contents or one of the pointers is `NULL`.

Example

The following code compares two pointers:

```
. . .
UCharT *ptr1 = NULL, *ptr2 = NULL;
. . .
if(F_PtrEqual(ptr1, ptr2, 256))
    F_Printf(NULL, "The pointers are equal to 256 chars.\n");
. . .
```

F_ReadBytes()

`F_ReadBytes()` reads a specified number of bytes from a channel to a buffer.

Synopsis

```
#include "fdetypes.h"
#include "fioutils.h"
. . .
IntT F_ReadBytes(PtrT ptr,
    IntT numBytes,
    ChannelT channel);
```

Arguments

`ptr`	A buffer to which to write the bytes
`numBytes`	The number of bytes to read
`channel`	The channel from which to read

Returns

The number of bytes read.

Example

The following code reads the first 255 bytes from a file located in the FrameMaker product directory:

```
. . .
UCharT buf[256];
ChannelT chan;
FilePathT *path;
IntT count;

path = F_PathNameToFilePath((StringT)"test.txt",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"r")) == NULL)
    return;
count = F_ReadBytes(buf, 255, chan);
buf[count] = '\0'; /* Add a NULL terminator. */
. . .
```

F_ReadLongs()

`F_ReadLongs()` reads a specified number of longs (four bytes) from a specified channel to a specified buffer. It swaps bytes if you have specified a byte order with `F_SetByteOrder()` or `F_ResetByteOrder()` and the byte order is different from the current platform's byte order.

Synopsis

```
#include "fdetypes.h"
#include "fioutils.h"
. . .
IntT F_ReadLongs(PtrT ptr,
    IntT num,
    ChannelT channel);
```

Arguments

`ptr`	A buffer to which to write
`num`	The number of longs to read
`channel`	The channel from which to read

Returns

The number of longs actually read.

Example

The following code reads and prints the first 256 long integers from a file located in the default directory:

```
. . .
IntT buf[256];
ChannelT chan;
FilePathT *path;
IntT count, i;

path = F_PathNameToFilePath((StringT)"test.dat",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"r")) == NULL) return;
count = F_ReadLongs((PtrT)buf, 256, chan);

for(i=0; i<count; i++)
   F_Printf(NULL, "%d ", buf[i]);
. . .
```

F_ReadShorts()

`F_ReadShorts()` reads a specified number of shorts (two bytes) from a specified channel to a specified buffer. It swaps bytes if you have specified a byte order with `F_SetByteOrder()` or `F_ResetByteOrder()` and the byte order is different from the current platform's byte order.

Synopsis

```
#include "fdetypes.h"
#include "fioutils.h"
. . .
IntT ReadShorts(PtrT ptr,
   IntT n,
   ChannelT channel);
```

Arguments

`ptr`	A buffer to which to write
`n`	The number of shorts to read
`channel`	The channel from which to read

Returns

The number of shorts actually read.

Example

The following code reads and prints the first 256 short integers from a file located in the default directory:

```
. . .
ChannelT chan;
ShortT buf[256];
FilePathT *path;
IntT count, i;

path = F_PathNameToFilePath((StringT)"test.dat",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"r")) == NULL) return;
count = F_ReadShorts((PtrT)buf, 256, chan);

for(i=0; i<count; i++)
   F_Printf(NULL, "%hd ", buf[i]);
. . .
```

F_Realloc()

`F_Realloc()` allocates a new block of memory and copies the contents of a specified block of memory to it. If the memory isn't available, `F_Realloc()` does not change the original block.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
PtrT F_Realloc(PtrT ptr,
   UIntT n,
   PUCharT flags);
```

Arguments

`ptr`	A pointer to the original block of memory
`n`	The number of bytes to allocate for the new block of memory
`flags`	Specifies whether to bail out (`DSE`) or return `NULL` (`NO_DSE`) if the memory you request isn't available

Returns

A pointer to the new block of memory, or `NULL` if it fails.

Example

Following are two functions to illustrate `F_Realloc()`. The first function uses `F_Realloc()` to increase the memory allocated to a pointer. It then clears the newly added memory space, and returns a pointer to the resulting memory. Note that it is important to clear the newly added memory; `F_Realloc()` does not clear it for you.

As an illustration, the second function invokes `growPointer()`. It first creates and prints a string of the characters "A - Z". After growing the pointer, it adds to the original string, and then prints the characters "A - Za - z".

```
PtrT growPtr(PtrT p, UIntT addBytes, UIntT contentSize)
{
    PtrT retp = NULL;

    p = F_Realloc(p, contentSize+addBytes, NO_DSE);
    if(!p)
        return(NULL);
    retp = p;
    p = (UByteT *)p + contentSize;
    F_ClearPtr(p, addBytes);
    return(retp);
}
```

```
VoidT testMem(VoidT) {
    UCharT *ptr = NULL, *start;
    UIntT i, len = 26;

    ptr = F_Alloc(len + 1, NO_DSE);
    if (ptr == NULL)
        return;
    F_ClearPtr(ptr, len);
    start = ptr;

    /* Make the first string of chars "A - Z" */
    for(i=0; i<len ;i++) {
        *ptr++ = ((UCharT)'A')+i;
    }
    *ptr = 0;
    F_Printf(NULL, "\n%s", (StringT)start);

    /* Make room for another 26 characters. */
    /* Test ptr to ensure growPtr succeded. */
    ptr = (UCharT *)growPtr(start, (UIntT)len, (UIntT)len+1);
    if(ptr == NULL)
        return;

    /* Append the chars "a - z" to the string */
    ptr += len;
    for(i=0; i<len;i++) {
        *ptr++ = ((UCharT)'a')+i;
    }
    *ptr = 0;
    F_Printf(NULL, "\n%s", (StringT)start);
    F_Free(start);
}
```

F_ReallocHandle()

`F_ReallocHandle()` allocates a new handle and a block of memory and copies the contents of a specified block of memory to it.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
HandleT F_ReallocHandle(HandleT handle,
    UIntT newSize,
    PUCharT flags);
```

Arguments

`handle`	A handle to the original block of memory
`newSize`	The number of bytes to allocate for the new block of memory
`flags`	Specifies whether to bail out `(DSE)` or return `NULL (NO_DSE)` if the memory you request isn't available

Returns

A handle to the new block of memory or `NULL` if it fails.

Example

The following code reallocates an additional 1K of memory to a handle:

```
. . .
HandleT hndl = NULL;

hndl = F_AllocHandle(66000, NO_DSE);
if(hndl == NULL) return;
if(!F_ReallocHandle(hndl, F_GetHandleSize(hndl)+1024, NO_DSE))
    F_Printf(NULL, "Couldn't reallocate handle.\n");
. . .
```

F_RenameFile()

`F_RenameFile()` renames a specified file or directory within a file system.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
ErrorT F_RenameFile(FilePathT *filepath,
   FilePathT *newfile);
```

Arguments

`filepath`	The filepath of the file or directory to rename
`newfile`	The new filepath

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails. If `newfile` specifies a different file system than `filepath`, `F_RenameFile()` fails. It also fails if the file or directory specified by `newfile` already exists, the parent directory doesn't exist, or permission is denied.

Example

The following code renames `t1.txt` to `t2.txt`:

```
. . .
FilePathT *filepath, *newfile;
filepath = F_PathNameToFilePath("<r><c>tmp<c>t1.txt",
                NULL, FDIPath);
newfile = F_PathNameToFilePath("<r><c>tmp<c>t2.txt",
                NULL, FDIPath);
if(F_RenameFile(filepath, newfile) != FdeSuccess)
   F_Printf(NULL, "Couldn't rename file.\n");
. . .
```

F_ResetByteOrder()

`F_ResetByteOrder()` sets the byte ordering for a specified channel to big-endian. Subsequent calls to FDE I/O functions, such as `F_ReadShorts()` and `F_WriteLongs()`, will swap bytes if the platform is little-endian.

Synopsis

```
#include "fdetypes.h"
#include "fioutils.h"
. . .
VoidT F_ResetByteOrder(ChannelT channel);
```

Arguments

`channel`	The channel for which to set the byte order

Returns

`VoidT`.

Example

The following code sets a channel's byte ordering to big-endian:

```
. . .
ChannelT chan;
. . .
F_ResetByteOrder(chan);
. . .
```

See also

"F_SetByteOrder()" on page 625.

F_ResetDirHandle()

`F_ResetDirHandle()` resets a file handle so that the next time you call `F_FilePathGetNext()`, it returns the first entry in the directory.

It is illegal to call `F_ResetDirHandle()` if the directory specified by `handle` is closed.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
ErrorT F_ResetDirHandle(DirHandleT handle);
```

Arguments

`handle`	The handle to reset. You must specify a handle returned by `F_FilePathOpenDir()`.

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

The following code resets the directory handle for the `tmp` directory.

```
. . .
DirHandleT handle;
FilePathT *path, *file;
IntT statusp;

path = F_PathNameToFilePath((StringT)"<r><c>tmp",
                    NULL, FDIPath);
handle = F_FilePathOpenDir(path, &statusp);
if(!handle) return;
file = F_FilePathGetNext(handle, &statusp);
. . .
F_ResetDirHandle(handle);
file = F_FilePathGetNext(handle, &statusp);
. . .
```

F_Scanf()

`F_Scanf()` reads formatted input from a specified channel. `F_Scanf()` is similar to `fscanf()`, except it doesn't support the `p` conversion character.

You can precede the conversion character (`d`, `o`, `x`, or `u`) with `l` or `h`. To specify `IntT`, precede the conversion character with `l`. To specify `ShortT`, precede it with `h`. By default, the conversion characters `d`, `o`, `x`, and `u` specify `IntT`.

IMPORTANT: This function only gives consistent results across multiple platforms when reading text in the `FrameRoman` encoding.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
IntT F_Scanf(ChannelT channel,
   StringT format, ...);
```

Arguments

`channel`	The channel from which to read input
`format`	The format for the input

Returns

The number of input items successfully matched and assigned.

Example

The following code reads formatted input and prints it:

```
. . .
IntT   i;
RealT fp;
FilePathT *path;
ChannelT chan;

F_Printf(NULL, "Enter an integer and floating-point:\n");
path = F_PathNameToFilePath((StringT)"test.txt",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"r")) == NULL) return;
F_Scanf(chan, "%d %f", &i, &fp);
F_Printf(NULL, "The numbers were: %d %f\n", i, fp);
. . .
```

F_SetAssert()

`F_SetAssert()` registers an assertion-handler function and returns the client's original assertion handle.

The FDK executes the assertion handler when a client calls `F_Assert()` and assertion failure occurs. After your assertion-handler function returns, the FDE's assertion handler is called to clean up the system and exit the client properly. It is illegal to call `abort()` or `exit()` from the assertion handler.

Synopsis

```
#include "fdetypes.h"
#include "fassert.h"
. . .
ProcedureT F_SetAssert(ProcedureT myHandler);
```

Arguments

`myHandler`	The assertion handler function

The assertion function `myHandler` is defined as:

```
VoidT myHandler();
```

Returns

`VoidT`.

Example

See "F_Assert()" on page 498.

F_SetByteOrder()

`F_SetByteOrder()` sets the byte ordering for a specified channel to little-endian. Subsequent calls to FDE I/O functions, such as `F_ReadShorts()` and `F_WriteLongs()`, will swap bytes if the platform is big-endian.

Synopsis

```
#include "fdetypes.h"
#include "fioutils.h"
. . .
VoidT F_SetByteOrder(ChannelT channel);
```

Arguments

`channel`	The channel for which to set byte ordering to little-endian

Returns

`VoidT`.

Example

The following code sets a channel's byte ordering to little-endian so that subsequent FDE I/O calls will swap bytes if the platform is big-endian:

```
. . .
ChannelT chan;
. . .
F_SetByteOrder(chan);
. . .
```

See also

"F_ResetByteOrder()" on page 620.

F_SetDSExit()

`F_SetDSExit()` sets the direct straight exit (DSE) function. The FDE calls this function when you call an FDE memory allocation function, such as `F_Alloc()`, with the `flags` argument set to `DSE`, and then memory allocation is denied. `F_SetDSExit()` returns the current DSE function.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
ProcedureT F_SetDSExit(ProcedureT fn);
```

Arguments

`fn`	The DSE function to set

The DSE function you specify for `fn` should clean up and exit the client. If the DSE function returns, the FDE aborts the client. By default, the DSE function is `NULL`; the FDE aborts the client when you call an FDE memory allocation function with `flags` set to `DSE` and memory allocation is denied.

Returns

The previous DSE function.

Example

The following code defines and sets a DSE function, and then makes a memory allocation call that calls the DSE function if there isn't enough memory:

```
. . .
VoidT DSExitFn(); /* Declaration */
. . .
VoidT DSExitFn()
{
   F_Warning((StringT) "Out of memory...Bailing out.\n");
   F_ApiBailOut();
}
. . .
UCharT *ptr = NULL;

F_SetDSExit(DSExitFn);
ptr = F_Alloc(10000000, DSE);
. . .
```

F_SetICUDataDir()

Sets the ICU data directory for the calling process.

NOTE: This function can accept the path in ASCII only. The path must be on a local, a mapped drive or a network path.

NOTE: ICU data directory is set on a per-process basis. Certain types of clients, like synchronous DLL clients on Windows, reside in the same process space as FrameMaker. Such clients do not need to set the ICU data directory since FrameMaker sets the ICU data directory for its process. If such clients set the ICU data directory incorrectly using this function, the entire FrameMaker process and other clients might get affected. Such clients should, therefore, be careful while setting the ICU data directory.

Synopsis

```
#include "fencode.h"

. . .

VoidT F_SetICUDataDir (ConStringT path);
```

Arguments

`path`	The ICU Data Directory path (absolute path, not relative)

Returns

`VoidT`

Example

The following code sets the ICU data directory of a client to the ICU data directory shipped with FrameMaker:

```
...
UCharT icu_path[256];
StringT icu_dir="\icu_data";
StringT fminit_path = F_ApiGetString(0, FV_SessionId,
FP_FM_InitDir);
UCharT *t,*s;
s=fminit_path;
while(*s)
*t++ = *s++;
s=icu_dir;
while(*s)
*t++ = *s++;
*t=0;
F_SetICUDataDir((ConStringT) icu_path);
...
```

F_Sprintf()

`F_Sprintf()` prints formatted output to a string. `F_Sprintf()` is similar to `sprintf()`, except it doesn't support the `p` conversion character.

All the arguments that you pass to `F_Sprintf()` must be FDE types and should be typecast.

You can precede the conversion character (`d`, `o`, `x`, or `u`) with `l` or `h`. To specify `IntT`, precede the conversion character with `l`. To specify `ShortT`, precede it with `h`. By default, the conversion characters `d`, `o`, `x`, and `u` specify `IntT`.

IMPORTANT: If you don't typecast the arguments you pass to `F_Sprintf()`, the output it generates may be different on different platforms.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
IntT F_Sprintf(StringT str,
   StringT format,
   ...);
```

Arguments

`str`	The string to which to print the formatted output
`format`	The formatted string to print

Returns

The length of the printed string.

Example

The following code uses `F_Sprintf()` to write characters to a buffer. It prints the following string:

```
String: FDK. Character: c. Int: 42. Real 3.141500
```

```
. . .
UCharT  buf[256];
IntT i;
i  = F_Sprintf(buf,   "String: %s. ", (StringT)"FDK");
i += F_Sprintf(buf+i, "Character: %c. ", (UCharT)'c');
i += F_Sprintf(buf+i, "Int: %d. ", (IntT)42);
i += F_Sprintf(buf+i, "Real: %f.", (RealT)3.1415);
F_Printf(NULL, "%s\n", buf);
. . .
```

F_Sscanf()

`F_Sscanf()` reads formatted input from a string. It is equivalent to `sscanf()`.

You can precede the conversion character (`d`, `o`, `x`, or `u`) with `l` or `h`. To specify `IntT`, precede the conversion character with `l`. To specify `ShortT`, precede it with `h`. By default, the conversion characters `d`, `o`, `x`, and `u` specify `IntT`.

IMPORTANT: This function only gives consistent results across multiple platforms when reading text in the `FrameRoman` encoding.

Synopsis

```
#include "fdetypes.h"
#include "futils.h"
. . .
IntT F_Sscanf(StringT str,
   StringT format,
   ...);
```

Arguments

`str`	The string from which to read input
`format`	The format of the input

Returns

The number of input items that successfully matched and assigned.

Example

The following code uses `F_Sscanf()` to read formatted input from a buffer. It prints the string:

```
The numbers are: 15 and 12.300000
```

```
. . .
IntT   i;
RealT fp;
static UCharT  buf[] = "15 12.3";
F_Sscanf(buf, "%d%f", &i, &fp);
F_Printf(NULL, "The numbers are: %d and %f\n", i, fp);
. . .
```

F_StrAlphaToInt()

`F_StrAlphaToInt()` converts an alphanumeric string into an integer.

IMPORTANT: With this function you can only use strings that contain characters in the 7-bit ASCII range.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrAlphaToInt(ConStringT s);
```

Arguments

`s`	The string to convert

Returns

The integer equivalent of the specified string.

Example

The following code prints the string `1000-100 = 900`:

```
. . .
F_Printf(NULL, "1000 - 100 = %d\n",
          F_StrAlphaToInt("1000") - (IntT)100);
. . .
```

F_StrAlphaToIntUnicode()

Converts a UTF-8 alphanumeric string into an integer

NOTE: This function won't work for numeric characters like Ⅳ (code point 0x2163) that aren't in a decimal system.This function can take decimal digits from mixed decimal systems (6 ४ , where ४ is 4 in devanagiri, is valid and interpreted as 64).

Synopsis

```
#include "fencode.h"

. . .

IntT F_StrAlphaToIntUnicode (ConStringT string);
```

Arguments

`s`	The string to convert

Returns

The integer equivalent of the specified string

Example

The following code prints ४२ + २४ = 66

```
...
StringT devanagiri_four="\xE0\xA5\xAA";
StringT devanagiri_two ="\xE0\xA5\xA8";
UCharT forty2[256];
UCharT twenty4[256];
IntT res;

FontEncIdT feId = F_FdeInitFontEncs((ConStringT)"UTF-8");
F_StrTruncEnc(forty2,0,feId);
F_StrTruncEnc(twenty4,0,feId);

F_StrCpy(forty2, devanagiri_four);
F_StrCat(forty2, devanagiri_two);
F_StrCpy(twenty4, devanagiri_two);
F_StrCat(twenty4, devanagiri_four);

res = F_StrAlphaToIntUnicode(forty2) +
F_StrAlphaToIntUnicode(twenty4);
F_Printf(NULL,"%s + %s=%d", fourPoint2, twoPoint4, res);
...
```

F_StrAlphaToReal()

`F_StrAlphaToReal()` converts an alphanumeric string to a `PRealT`.

IMPORTANT: With this function you can only use strings that contain characters in the 7-bit ASCII range.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
PRealT F_StrAlphaToReal(ConStringT s);
```

Arguments

`s`	The string to convert

Returns

A `PRealT` corresponding to the specified string.

Example

The following code prints the string `10.05-5.05 = 5.00`:

```
. . .
F_Printf(NULL, "10.05 - 5.05 = %2.2f\n",
          F_StrAlphaToReal("10.05") - (RealT)5.05);
. . .
```

F_StrAlphaToRealUnicode()

Converts a UTF-8 alphanumeric string into a PRealT

NOTE: This function won't work for numeric characters like Ⅳ (code point 0x2163) that aren't in a decimal system. This function can take decimal digits from mixed decimal systems (6.४, where ४ is 4 in devanagiri, is valid and interpreted as 6.4).

Synopsis

```
#include "fencode.h"

. . .

PRealT F_StrAlphaToRealUnicode (ConStringT string);
```

Arguments

`string`	The string to convert

Returns

The PRealT corresponding to the specified string

Example

The following code prints ४ . २ + २ . ४ = 6.6

```
...
StringT devanagiri_four="\xE0\xA5\xAA";
StringT devanagiri_two ="\xE0\xA5\xA8";
UCharT forty2[256];
UCharT twenty4[256];
PRealT res;

FontEncIdT feId = F_FdeInitFontEncs((ConStringT)"UTF-8");
F_StrTruncEnc(forty2,0,feId);
F_StrTruncEnc(twenty4,0,feId);

F_StrCpy(forty2, devanagiri_four);
F_StrCat(forty2, ".");
F_StrCat(forty2, devanagiri_two);
F_StrCpy(twenty4, devanagiri_two);
F_StrCat(twenty4, ".");
F_StrCat(twenty4, devanagiri_four);

res = F_StrAlphaToRealUnicode(forty2)+
F_StrAlphaToRealUnicode(twenty4);
F_Printf(NULL,"%s + %s=%f", fourPoint2, twoPoint4, res);
...
```

F_StrBrk()

`F_StrBrk()` returns a pointer to the first occurrence in `s1` of any character in `s2`.

IMPORTANT: This function only works with characters that use the FrameRoman encoding. This is important if your document includes Asian text.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
StringT F_StrBrk(StringT s1,
    ConStringT s2);
```

Arguments

`s1`	The string to search
`s2`	The characters to search for

Returns

A pointer to the first occurrence in `s1` of any character in `s2`, or `NULL` if none of the characters in `s2` occurs in `s1`.

Example

The following code prints the string `Earthshaker`:

```
. . .
StringT s1, s;
s1 = F_StrCopyString("Poseidon Earthshaker");
s =  F_StrBrk(s1, (StringT)"pOE");
if (s != NULL) F_Printf(NULL, "%s\n", s);
. . .
```

F_StrBrkUTF8 ()

Returns a pointer to the first occurrence in UTF-8 string s1 of any UTF-8 character in s2.

Synopsis

```
#include "fencode.h"

. . .

StringT F_StrBrkUTF8 (StringT s1, ConStringT s2);
```

Arguments

`s1`	The string to search
`s2`	A string containing characters to search for

Returns

A pointer to the first occurrence in s1 of any character in s2, or NULL if none of the characters in s2 occurs in s1.

Example

The following code prints the string 1 ⊕ 2 ⊖ 3 - ⊕ 2 ⊖ 3 - ⊖ 3

```
. . .
StringT s, s1;
F_FdeInitFontEncs("UTF-8");

s1 = F_StrCopyString("1 \xE2\x8A\x95 2 \xE2\x8A\x96 3");
F_Printf(NULL, s1);
s = F_StrBrkUTF8(s1, "\xE2\x8A\x95")
F_Printf(NULL, " - %s",s);
s = F_StrBrkUTF8(s1, "\xE2\x8A\x96")
F_Printf(NULL, " - %s",s);
...
```

F_StrCat()

`F_StrCat()` concatenates two strings, terminating the resulting string with a null character.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrCat(StringT s1,
    ConStringT s2);
```

Arguments

`s1`	The first string to concatenate
`s2`	The string to append to the end of `s1`

`F_StrCat()` appends the contents of `s2` to `s1`. For it to work correctly, you must allocate additional memory to `s1`.

Returns

The final length of the concatenated string.

Example

The following code prints the string `It was the best of times. It was the worst of times.`:

```
. . .
StringT s1, s2;
s1 = F_StrCopyString("It was the best of times. ");
s2 = F_StrCopyString("It was the worst of times.");
s1 = F_Realloc(s1, F_StrLen(s1)+F_StrLen(s2)+1, NO_DSE);
F_StrCat(s1, s2);
F_Printf(NULL, "%s\n", s1);
. . .
```

F_StrCatCharN()

`F_StrCatCharN()` appends a character to a string, limiting the length of the resulting string to a specified length.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrCatCharN(StringT s1,
    PUCharT c,
    UIntT n);
```

Arguments

`s1`	The string
`c`	The character to append to the string
`n`	The length (including the terminating `0`) to which to limit the concatenated string

Returns

The final length of the concatenated string.

Example

The following code prints the string `Cronides.`:

```
. . .
StringT s;
s = F_StrNew(F_StrLen("Cronides")+(IntT)1);
F_StrCpy(s, (StringT)"Cronides");
F_StrCatCharN(s,(UCharT)'.', (IntT)10);
F_Printf(NULL, "%s\n", s);
. . .
```

F_StrCatDblCharNEnc()

`F_StrCatDblCharNEnc()` appends a double-byte character to a string, limiting the length of the resulting string to a specified length.

IMPORTANT: This function specifies string length in terms of bytes and not characters. Also, `n` includes the terminating byte. A value of 7 for `n` limits the length of the string to three double-byte characters.

Synopsis

```
#include "fencode.h"
. . .
IntT F_StrCatDblCharN(StringT s1,
   UCharT first,
   UCharT second,
   UIntT n,
   FontEncIdT feId);
```

Arguments

`s1`	The string
`first`	The first byte of the double-byte character to append to the string
`second`	The second byte of the double-byte character to append to the string
`n`	The length (including the terminating `NULL` byte) to which to limit the concatenated string
`feId`	The ID of the font encoding for the charcter you are appending to the string

Returns

The final length in bytes of the concatenated string.

Example

The following code prints the string s, with the character 82CD added to the end of it:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
StringT s, myString;
. . .
/* Assuming a string of double-byte chars in myString... */
s = F_StrNew(F_StrLen(myString)+(IntT)2);
F_StrCpy(s, myString);
F_StrCatDblCharNEnc(s,\x82, \xCD, F_StrLen(s), feId);
F_Printf(NULL, "%s\n", s);
. . .
```

F_StrCatIntN()

`F_StrCatIntN()` converts an integer to a string and appends it to a string, limiting the length of the resulting string to a specified length.

IMPORTANT: This function specifies string length in terms of bytes and not characters. Also, `n` includes the terminating byte. This is important if the string contains double-byte characters.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrCatIntN(StringT s1,
    IntT i,
    UIntT n);
```

Arguments

`s1`	The string
`i`	The integer to append to the string
`n`	The length (including the terminating `0`) to which to limit the concatenated string

Returns

The final length of the concatenated string.

Example

The following code prints the string `Number 9`:

```
. . .
StringT s;
s = F_StrCopyString("Number ");
s = F_Realloc(s, F_StrLen(s)+(IntT)1, NO_DSE);
F_StrCatIntN(s, (IntT)9, (IntT)9);
F_Printf(NULL, "%s\n", s);
. . .
```

F_StrCatN()

`F_StrCatN()` concatenates two strings, limiting the length of the resulting string to a specified length.

IMPORTANT: This function specifies string length in terms of bytes and not characters. Also, `n` includes the terminating byte. This is important if the string contains double-byte characters. For strings in an encoding other than `FrameRoman`, use `F_StrCatNEnc()`.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrCatN(StringT s1,
   ConStringT s2,
   UIntT n);
```

Arguments

`s1`	The first string to concatenate
`s2`	The string to append to `s1`
`n`	The length (including the terminating `0`) to which to limit the concatenated string

`F_StrCat()` appends the contents of `s2` to `s1`. For it to work correctly, you must allocate additional memory to `s1`.

Returns

The final length of the concatenated string.

Example

The following code prints the string `The bitter wrath of Achilleus, lord of the house`:

```
. . .
StringT s1, s2;
s1 = F_StrCopyString("The bitter wrath of Achilleus, ");
s2 = F_StrCopyString("lord of the house of Peleus");
s1 = F_Realloc(s1, F_StrLen(s1)+(IntT)18, NO_DSE);
F_StrCatN(s1, s2, F_StrLen(s1)+(IntT)18);
F_Printf(NULL, "%s\n", s1);
. . .
```

F_StrCatNEnc()

`F_StrCatNEnc()` concatenates two strings of the specified encoding, limiting the length of the resulting string to a specified length.

IMPORTANT: This function specifies string length in terms of bytes and not characters. Note that a string of three double-byte characters has three characters and six bytes.

Synopsis

```
#include "fencode.h"
. . .
IntT F_StrCatNEnc(StringT s1,
    ConStringT s2,
    UIntT n,
    FontEncIdT feId);
```

Arguments

`s1`	The first string to concatenate
`s2`	The string to append to `s1`
`n`	The length (including the terminating `0`) to which to limit the concatenated string
`feId`	The ID of the font encoding for your strings

`F_StrCat()` appends the contents of `s2` to `s1`. For it to work correctly, you must allocate sufficient memory to `s1`.

Returns

The final length in bytes of the concatenated string.

Example

The following code prints s1 after concatenating at least 9 characters of s2 to the end of s1. Note that `F_StrCatNEnc()` counts by bytes. With Japanese text, characters could be either one or two bytes each. For this reason, don't use `F_StrLenEnc()` which returns the string length in characters; instead, use `F_StrLen()` to return the string length in bytes.

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
IntT i;
StringT s1, s2;
/*
* Assuming you have Japanese text in s1 and s2... */
i = F_StrLen(s1) + (F_StrLen(s2) + 1);
if (s1 = F_Realloc(s1, i, NO_DSE)) {
   F_StrCatNEnc(s1, s2, i, feId);
   F_Printf(NULL, "%s\n", s1);
}
. . .
```

F_StrCatUTF8CharNByte ()

Appends a UTF8 character to a string, limiting the length of the resulting string to a specified length

IMPORTANT: The length of the resulting string is calculated in bytes, not characters. This is important because a UTF-8 character may take up to 4 bytes.

Synopsis

```
#include "fencode.h"

. . .

IntT F_StrCatUTF8CharNByte ( StringT s, const UCharT *c, IntT
n);
```

Arguments

`s`	The string
`c`	The character to append
`n`	The length limit (in bytes) for `s1`

Returns

Final length of the copied string in terms of bytes

Example

The following code prints the string Бо + г = Бог

```
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
UCharT s[256];
StringT a = "\xD0\x91\xD0\xBE";
StringT b = "\xD0\xB3";

s[0]=0;
F_StrCpy(s,a);
F_StrCatUTF8CharNByte( s, b, 256);
F_Printf(NULL, "%s + %s = %s\n", a, b, s);
. . .
```

F_StrChr()

`F_StrChr()` finds the first occurrence of a character in a string.

IMPORTANT: This function only works with characters that use the `FrameRoman` encoding. If your document includes Asian text, use `F_StrChrEnc()` instead.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
StringT F_StrChr(StringT s, PUCharT c);
```

Arguments

`s`	The string to search
`c`	The character to search `s` for

Returns

A pointer to the first occurrence of `c` in `s`, or `NULL` if `c` does not occur in `s`.

Example

The following code prints the string `Hecuba?`:

```
. . .
StringT s, string;
s = F_StrCopyString("Who was he to Hecuba?");
string = F_StrChr(s, 'H');
F_Printf(NULL, "%s\n", string);
. . .
```

F_StrChrEnc()

`F_StrChrEnc()` finds the first occurrence in a string, of a double-byte character of the specified encoding.

Synopsis

```
#include "fencode.h"
. . .
StringT F_StrChrEnc(ConStringT s,
   UCharT first,
   UCharT second,
   FontEncIdT feId);
```

Arguments

`s`	The string to search
`first`	The first byte of the character to search `s` for
`second`	The second byte of the character to search `s` for
`feId`	The ID of the font encoding to check against

Returns

A pointer to the first occurrence in `s` of the double byte character represented by `first` and `second`, or `NULL` if such a character does not occur in `s`.

Example

The following code returns a string beginning with the Japanese character 89C3. Also see the example for "F_StrChr()" on page 645:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
StringT s, string;

/* Assuming a string of double-byte chars in s... */
string = F_StrChrEnc(s, \x89, \xC3, feId);
. . .
```

F_StrChrUTF8()

Returns a pointer to the first occurrence of a UTF-8 character in a UTF-8 string, starting from the end of the string

Synopsis

```
#include "fencode.h"
. . .
StringT F_StrChrUTF8(ConStringT s, const UCharT *c);
```

Arguments

s	The string to search
c	The character (in terms of a byte sequence) to search s for

Returns

A pointer to the first occurrence of c in s, or NULL if c doesn't occur in s

Example

The following code prints "Бог и взял". Here the long hex sequence "\xD0\x91...\xD0\xBB" is the representation of "Бог дал, Бог и взял" in UTF-8 and "\xD0\x91" of the character 'Б'

```
#include "fencode.h"
. . .
StringT s, c, string;
F_FdeInitFontEncs((ConStringT)"UTF-8");

s =
F_StrCopyString("\xD0\x91\xD0\xBE\xD0\xB3\x20\xD0\xB4\xD0\xB0\xD
0\xBB\x2C\x20\xD0\x91\xD0\xBE\xD0\xB3\x20\xD0\xB8\x20\xD0\xB2\xD
0\xB7\xD1\x8F\xD 0\xBB");

c = "\xD0\xBB";

string = F_StrRChrUTF8(s, c);
F_Printf(NULL, "%s\n", string);
. . .
```

F_StrCmp()

`F_StrCmp()` is equivalent to `strcmp()`. It compares the ASCII value of each character in two null-terminated byte strings.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrCmp(ConStringT s1,
    ConStringT s2);
```

Arguments

`s1`	A string
`s2`	A string to compare to `s1`

Returns

An integer greater than `0` if `s1` is lexicographically greater than `s2`; `0` if the strings are equal; or an integer less than 0 if `s1` is less than `s2`.

Example

The following code prints the string `The strings compare`:

```
. . .
if(!F_StrCmp((StringT)"string1", (StringT)"string1"))
    F_Printf(NULL, "The strings compare\n");
. . .
```

F_StrCmpN()

`F_StrCmpN()` compares two strings to a specified number of characters.

IMPORTANT: This function only works with characters that use the `FrameRoman` encoding. If your document includes Asian text, use `F_StrCmpNEnc()` instead.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrCmpN(ConStringT s1,
   ConStringT s2,
   IntT n);
```

Arguments

`s1`	A string
`s2`	A string to compare to `s1`
`n`	The number of characters to compare

Returns

An integer greater than `0` if `s1` is lexicographically greater than `s2`; `0` if the strings are equal; or an integer less than 0 if `s1` is less than `s2`.

Example

The following code prints the string `The strings compare`:

```
. . .
if(!F_StrCmpN((StringT)"string1", (StringT)"string2",(IntT) 6))
       F_Printf(NULL, "The strings compare.\n");
. . .
```

F_StrCmpNEnc()

`F_StrCmpNEnc()` compares two strings of the specified encoding up to a specified number of bytes. Note that case is not applicable for double-byte characters, so this function only considers case within the 7 bit ASCII range of characters.

Synopsis

```
#include "fencode.h"
. . .
IntT F_StrCmpNEnc(ConStringT s1,
    ConStringT s2,
    IntT n,
    FontEncIdT feId);
```

Arguments

`s1`	A string
`s2`	A string to compare to `s1`
`n`	The length of the strings, in bytes, by which to compare the strings
`feId`	The ID of the font encoding to check against

Returns

An integer greater than `0` if `s1` is lexicographically greater than `s2`; `0` if the strings are equal; or an integer less than 0 if `s1` is less than `s2`.

Example

The following code determines whether `s1` is lower in sort order than `s2`:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
ConStringT s1, s2;

. . .
/* Assuming you have Japanese text in string1 and string2... */
if(F_StrCmpNEnc(s1, s2, (IntT) 6, feId) < 0)
    F_Printf(NULL, "The first 6 bytes of s1 are lower in
        sort order than the first 6 bytes of s2.\n");
. . .
```

F_StrICmpNUTF8Char

Compares the Unicode code points of each character in two UTF-8 strings up to a specified number of UTF-8 characters (not bytes). This function ignores cases.

Synopsis

```
#include "fencode.h"

. . .

IntT F_StrICmpNUTF8Char ( ConStringT s1, ConStringT s2, IntT n);
```

Arguments

`s1`	A string
`s2`	A string to compare to `s1`

Returns

An integer greater than 0 if `s1` is lexicographically greater than `s2`; 0 if the strings are equal; or an integer less than 0 if `s1` is less than `s2`.

Example

The following code prints the string `First 2 chars of Бог and Боо are equal`

```
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
StringT s1 = F_StrCopyString("\xD0\x91\xD0\xBE\xD0\xB3");
StringT s2 = F_StrCopyString("\xD0\x91\xD0\xBE\xD0\xBE");
if(!F_StrICmpNUTF8Char(s1, s2, 2))
   F_Printf(NULL, "First 2 chars of %s and %s are equal\n", s1,
s2);
. . .
```

F_StrCmpUTF8()

Compares the Unicode code point of each character in two null-terminated UTF-8 strings.

Synopsis

```
#include "fencode.h"

. . .

IntT F_StrCmpUTF8(ConStringT s1, ConStringT s2);
```

Arguments

`s1`	A string
`s2`	A string to compare to `s1`

Returns

An integer greater than 0 if `s1` is lexicographically greater than `s2`; 0 if the strings are equal; or an integer less than 0 if `s1` is less than `s2`

Example

The following code prints the string `The strings ÁÎã and ÁÎã are equal`

```
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
StringT s1 = F_StrCopyString("\xD0\x91\xD0\xBE\xD0\xB3");
StringT s2 = F_StrCopyString("\xD0\x91\xD0\xBE\xD0\xB3");
if(!F_StrCmp(s1, s2))
F_Printf(NULL, "The strings %s and %s are equal\n", s1, s2);
. . .
```

The following code prints the string Бог дал `is less than` Бог и взял

```
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");

StringT s1 =
F_StrCopyString("\xD0\x91\xD0\xBE\xD0\xB3\x20\xD0\xB4\xD0\xB0\xD
0\xBB");

StringT s2 =
F_StrCopyString("\xD0\x91\xD0\xBE\xD0\xB3\x20\xD0\xB8\x20\xD0\xB
2\xD0\xB
7\xD1\x8F\xD0\xBB");

if(F_StrCmp(s1, s2)<0)
F_Printf(NULL, "%s is less than %s\n", s1, s2);
else
F_Printf(NULL, "%s is greater than %s\n", s1, s2);
. . .
```

F_StrCmpUTF8Locale()

Compares the UTF-8 string based on Unicode Collation Algorithm (UCA)

NOTE: Because the comparison honors the current system locale, it varies slightly with language rules in different locales.

This doesn't perform a code point-based comparison, but instead uses UCA and localization information in order to compare strings correctly. For example, if used for sorting, it sorts all digits together, sorting the Devanagiri representation of 2 between the English representations of 1 and 3. Similarly, the 'Double Width B' sorts between 'A' and 'C' even though its code point is much higher. This also performs canonical normalization to ensure that the Unicode character sequences `0x00C4` and `0x0041 0x0308`, which are both ways of representing Ä, are considered equivalent.

Synopsis

```
#include "fencode.h"

. . .

IntT F_StrCmpUTF8Locale (ConStringT s1, ConStringT s2);
```

Arguments

`s1`	A string
`s2`	A string to compare to s1

Returns

An integer greater than 0 if `s1` is lexicographically greater than `s2`; 0 if the strings are equal; or an integer less than 0 if `s1` is less than `s2`.

Example

The following code prints the string `a<B<c`

```
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
StringT B = "\xEF\xBC\xA2";/* double width B */

if ((F_StrICmpUTF8Locale("a", B)<0)
   && (F_StrICmpUTF8Locale(B, "c")<0))
       F_Printf(NULL, "a<B<c");
. . .
```

F_StrICmpUTF8Locale()

Compares the UTF-8 string based on Unicode Collation Algorithm (UCA) case insensitively

NOTE: Because the comparison honors the current system locale, it varies slightly with language rules in different locales.

This doesn't perform a code point-based comparison, but instead uses UCA and localization information in order to compare strings correctly. For example, if used for sorting, it sorts all digits together, sorting the Devanagiri representation of 2 between the English representations of 1 and 3. Similarly, the 'Double Width B' sorts between 'a' and 'C' even though its code point is much higher. This also performs canonical normalization to ensure that the Unicode character sequences 0x00C4 and 0x0041 0x0308, which are both ways of representing Ä, are considered equivalent.

F_StrConvertEnc(), F_StrConvertEnc_IgnoreControlChars(), F_StrConvertEnc_ConvertControlChars()

Utility function for conversion between various encodings.

Synopsis

```
#include "fencode.h"

. . .

StringT F_StrConvertEnc (ConStringT in,
                    FTextEncodingT from, FTextEncodingT to);

StringT F_StrConvertEnc_IgnoreControlChars (ConStringT in,
                    FTextEncodingT from, FTextEncodingT to);

StringT F_StrConvertEnc_ConvertControlChars (ConStringT in,
                    FTextEncodingT from, FTextEncodingT to);
```

`F_StrConvertEnc` is identical to `F_StrConvertEnc_IgnoreControlChars`. `F_StrConvertEnc_IgnoreControlChars` doesn't modify the lower 32 characters of `0x00-0x1F` and copies them byte-by-byte irrespective of from and to encoding (except in the case of conversion between UTF-16 and another encoding, where `0x0010` is converted to `0x10` and vice-versa).

`F_StrConvertEnc_ConvertControlChars` does modify the lower 32 characters of `0x00-0x1F` assuming them to be in the from encoding. Thus it converts `0x14` to `0x2003` if `from` is `F_EncMakerRoman` and `to` is `F_EncUTF8`.

UTF-16 strings are expected in the endianness of the platform and are returned in the platform endianness. You must ensure that the input string in is valid in the encoding from. Incorrect or lossy conversions due to incorrect input strings, incorrect encodings, and conversion limitations inherent to the encodings might cause question marks '?' to appear in the string. An empty or `NULL` string might also be returned. For incorrect UTF-8 inputs (when from is `F_EncUTF8`), a string with 3 question marks "`???`" is returned.

NOTE: Strings returned by this function must be freed by using `F_Free`

NOTE: If from is `F_EncUTF16`, the `UChar16T` pointer pointing towards the UTF-16 string should be typecasted to `ConStringT` (it would be treated correctly inside). If to is `F_EncUTF16`, the return value should be typecasted to `UChar16T *`. `F_EncSpecialSymbol`, `F_EncSpecialZapfDingbats` and `F_EncSpecialWingdings` can only be used as from encodings.

Arguments

`in`	The string to convert
from	The encoding from which to convert
`to`	The encoding to which the string must be converted

The possible values of the `FTextEncodingT` parameters **`from`** and `to` are:

F_EncMakerRoman	Corresponds to `FrameRoman` encoding
F_EncISOLatin1	
F_EncASCII	
F_EncANSI	
F_EncMacASCII	
F_EncJIS7	
F_EncShiftJIS	Corresponds to `JISX0208.ShiftJIS` encoding
F_EncJIS8_EUC	
F_EncBig5	Corresponds to `BIG5` encoding
F_EncCNS_EUC	
F_EncGB8_EUC	Corresponds to `GB2312-80.EUC` encoding
F_EncHZ	
F_EncKSC8_EUC	Corresponds to `KSC5601-1992` encoding
F_EncUTF8	

F_EncUTF16	
F_EncSpecialSymbol	Can only be used as `from` encoding
F_EncSpecialZapfDingbats	Can only be used as `from` encoding
F_EncSpecialWingdings	Can only be used as `from` encoding

Returns

The converted string

Example

The following code prints `Symbol` α β χ `Shift-JIS` あぶい

```
#include "fencode.h"
. . .
StringT symb="abc";
StringT sjis="\x82\xA0\x82\xD4\x82\xA2";
StringT temp;
F_FdeInitFontEncs((ConStringT)"UTF-8");
F_Printf(NULL, "Symbol ");
temp = F_StrConverEnc(symb, F_EncSpecialSymbol, F_EncUTF8);
F_Printf(NULL, temp);
F_Free(temp);
F_Printf(NULL, " Shift-JIS");
temp = F_StrConverEnc(sjis, F_EncShiftJIS, F_EncUTF8);
F_Printf(NULL, temp);
F_Free(temp);
...
```

F_StrCopyString()

`F_StrCopyString()` returns a copy of a specified string.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
StringT F_StrCopyString(ConStringT s);
```

Arguments

`s`	The string to copy

Returns

A copy of the specified string, or `NULL` if it fails. When you are finished with the string, use `F_Free()` to free it. For more information, see "F_Free()" on page 559.

Example

The following code creates a copy of the string `Doe-eyed Hera`. It prints the copied string and then deallocates it:

```
. . .
StringT s;
s = F_StrCopyString((StringT) "Doe-eyed Hera");
if(s != NULL)
{
    F_Printf(NULL, "%s\n", s);
    F_Free(s);
}
. . .
```

F_StrCpy()

`F_StrCpy()` copies a string to another string.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
VoidT F_StrCpy(StringT s1,
   ConStringT s2);
```

Arguments

`s1`	The string to copy `s2` to. It must point to a block of modifiable memory of the same size as `s2`.
`s2`	The string to copy.

`F_StrCpy()` does not automatically allocate memory to `s1`. For it to work correctly, you must allocate memory to it.

Returns

`VoidT`.

Example

The following code copies the string `Rosy-fingered dawn`:

```
. . .
StringT s1;
s1 = F_StrNew(F_StrLen("Rosy-fingered dawn")+1);
F_StrCpy(s1, (StringT) "Rosy-fingered dawn");
. . .
```

F_StrCpyN()

`F_StrCpyN()` copies a string to another string, limiting the target string to a specified number of bytes.

IMPORTANT: The length of the resulting string is calculated in bytes, not characters. This is important if the string contains double-byte characters.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrCpyN(StringT s1,
    ConStringT s2,
    UIntT n);
```

Arguments

`s1`	A string
`s2`	A string to copy to `s1`
`n`	The number of bytes to which to limit the length of `s1` (including the terminating `0`)

`F_StrCpyN()` does not automatically allocate memory to `s1`. For it to work correctly, you must allocate memory to it.

Returns

The final length of the copied string.

Example

The following code prints the string `Aias`:

```
. . .
StringT s1;
IntT len;
s1 = F_StrNew(5);
len = F_StrCpyN(s1, (StringT)"Aias Telemonian", (IntT)5);
F_Printf(NULL, "%s\n", s1);
. . .
```

F_StrCpyNEnc()

`F_StrCpyNEnc()` copies a string of the specified encoding to another, limiting the length of the target string to a specified number of bytes.

IMPORTANT: This function specifies string length in terms of bytes and not characters. Note that a string of three double-byte characters has three characters and six bytes.

Synopsis

```
#include "fencode.h"
. . .
IntT F_StrCpyNEnc(StringT s1,
   ConStringT s2,
   UIntT n,
   FontEncIdT feId);
```

Arguments

`s1`	A string
`s2`	The string to copy to `s1`
`n`	The length in bytes (including the terminating `0`) to which to limit the concatenated string
`feId`	The ID of the font encoding for your strings

`F_StrCpyNEnc()` copies the contents of `s2` to `s1`. For it to work correctly, you must allocate sufficient memory to `s1`.

Returns

The final length in bytes of the copied string.

Example

The following code prints the number of characters and bytes in s1, followed by the contents of s1. Also see “F_StrCpyN()” on page 660

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
IntT i, byteLen, charLen;
StringT s1, s2;

/* Assuming you have Japanese text in s2, */
/* when allocating the string it's safest to assume */
/* 2 bytes per char, plus the terminating byte. */
s1 = F_StrNew((IntT)9);
byteLen = F_StrCpyNEnc(s1, s2, (IntT)8, feId);
charLen = F_StrLenEnc(s1);
F_Printf(NULL, "These %d characters take up %d bytes:\n
       %s\n", charLen, byteLen, s1);
. . .
```

F_StrCpyNUTF8Char ()

Copies a string to another string, limiting the target string to a specified number of characters

IMPORTANT: The length of the resulting string is calculated in characters, not bytes. This is important because a UTF-8 character may take up to 4 bytes. The resulting string must have adequate space.

Synopsis

```
#include "fencode.h"

. . .

IntT F_StrCpyNUTF8Char (StringT s1,ConStringT s2,IntT n);
```

Arguments

`s1`	A string
`s2`	A string to copy to `s1`
`n`	The length limit (in characters) for `s1`

Returns

Final length of the copied string in terms of characters

Example

The following code prints the string `The string Бо was copied from Бог`

```
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
StringT s2 = F_StrCopyString("\xD0\x91\xD0\xBE\xD0\xB3");
UCharT s1[256];
s1[0]=0;
F_StrCpyNUTF8Char(s1,s2,2);
F_Printf(NULL, "The string %s was copied from %s\n", s1, s2);
. . .
```

F_StrEqual()

`F_StrEqual()` compares two strings.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
BoolT F_StrEqual(ConStringT s1,
    ConStringT s2);
```

Arguments

`s1`	A string
`s2`	A string to compare with `s1`

Returns

`True` if the strings are equal; otherwise, `False`.

Example

The following code prints the string `The strings are equal`:

```
. . .
if (F_StrEqual((StringT)"Myrmidon", (StringT)"Myrmidon"))
    F_Printf(NULL, "The strings are equal.\n");
. . .
```

F_StrEqualN()

`F_StrEqualN()` compares two strings up to a specified number of characters.

IMPORTANT: The number of characters to compare is calculated in bytes, not characters. This is important if the string contains double-byte characters.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
BoolT F_StrEqualN(ConStringT s1,
   ConStringT s2,
   UIntT n);
```

Arguments

`s1`	A string
`s2`	A string to compare with `s1`
`n`	The number of characters to which to compare the strings

Returns

`True` if the strings are equal to `n` characters; otherwise, `False`.

Example

The following code prints the string `The first 6 chars are equal`:

```
. . .
if(F_StrEqualN((StringT)"string1", (StringT)"string2",(IntT) 6))
   F_Printf(NULL, "The first 6 chars are equal.\n");
. . .
```

F_StrICmp()

`F_StrICmp()` compares two strings, ignoring case.

IMPORTANT: This function only works with characters that use the FrameRoman encoding. This is important if your document includes Asian text.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrICmp(ConStringT s1,
    ConStringT s2);
```

Arguments

`s1`	A string
`s2`	A string to compare to `s1`

Returns

An integer greater than `0` if `s1` is lexicographically greater than `s2`; `0` if the strings are equal; or an integer less than 0 if `s1` is less than `s2`.

Example

The following code prints the string `The strings compare`:

```
. . .
if(!F_StrICmp((StringT)"String1", (StringT)"string1"))
    F_Printf(NULL, "The strings compare.\n");
. . .
```

F_StrICmpEnc()

`F_StrICmpEnc()` compares two strings of the specified encoding, ignoring case. Note that double-byte characters have no case; when comparing strings, this function ignores the case of any characters in those strings that are within the 7-bit ASCII range.

Synopsis

```
#include "fencode.h"
. . .
IntT F_StrICmpEnc(ConStringT s1,
    ConStringT s2,
    FontEncIdT feId);
```

Arguments

`s1`	A string
`s2`	A string to compare to `s1`
`feId`	The ID of the font encoding to check against

Returns

An integer greater than `0` if `s1` is lexicographically greater than `s2`; `0` if the strings are equal; or an integer less than 0 if `s1` is less than `s2`.

Example

The following code determines whether to print the string, `"s1 is lower in sort order than s2."`:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
ConStringT s1, s2;

. . .
/* Assuming you have Japanese text in string1 and string2... */
if(F_StrICmpEnc(s1, s2, feId) < 0))
   F_Printf(NULL, "s1 is lower in sort order than s2.\n");
. . .
```

F_StrICmpN()

`F_StrICmpN()` compares two strings to a specified number of characters, ignoring case.

IMPORTANT: This function only works with characters that use the FrameRoman encoding. This is important if your document includes Asian text.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrICmpN(ConStringT s1,
   ConStringT s2,
   IntT n);
```

Arguments

`s1`	A string
`s2`	A string to compare to `s1`
`n`	The number of characters to compare

Returns

An integer greater than `0` if `s1` is lexicographically greater than `s2`; `0` if the strings are equal; or an integer less than 0 if `s1` is less than `s2`.

Example

The following code prints the string `The strings compare to six characters`:

```
. . .
if(!F_StrICmpN((StringT)"String1",(StringT)"string2", (IntT) 6))
   F_Printf(NULL, "The strings compare to six characters.\n");
. . .
```

F_StrICmpNEnc()

`F_StrICmpNEnc()` compares two strings of the specified encoding up to a specified number of bytes. This function ignores case for characters in the 7-bit ASCII range.

IMPORTANT: This function specifies string length in terms of bytes and not characters. Note that a string of three double-byte characters has three characters and six bytes.

Synopsis

```
#include "fencode.h"
. . .
IntT F_StrICmpNEnc(ConStringT s1,
   ConStringT s2,
   IntT n,
   FontEncIdT feId);
```

Arguments

`s1`	A string
`s2`	A string to compare to `s1`
`n`	The length of the strings, in bytes, by which to compare the strings
`feId`	The ID of the font encoding to check against

Returns

An integer greater than `0` if `s1` is lexicographically greater than `s2`; `0` if the strings are equal; or an integer less than 0 if `s1` is less than `s2`.

Example

The following code determines whether `s1` is lower in sort order than `s2`:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
ConStringT s1, s2;

. . .
/* Assuming you have Japanese text in string1 and string2... */
if(F_StrICmpNEnc(s1, s2, (IntT) 6, feId) < 0))
   F_Printf(NULL, "The first 6 bytes of s1 are lower in
      sort order than the first 6 bytes of s2.\n");
. . .
```

F_StrIEqual()

`F_StrIEqual()` compares two strings, ignoring case.

IMPORTANT: This function only works with characters that use the FrameRoman encoding. This is important if your document includes Asian text.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
BoolT F_StrIEqual(ConStringT s1,
   ConStringT s2);
```

Arguments

`s1`	A string
`s2`	A string to compare with `s1`

Returns

`True` if the strings are equal; otherwise, `False`.

Example

The following code prints the string `The strings are equal`:

```
. . .
if (F_StrIEqual((StringT)"string", (StringT)"String"))
   F_Printf(NULL, "The strings are equal.\n");
. . .
```

F_StrIEqualEnc()

`F_StrIEqualEnc()` compares two strings of the specified encoding. This function ignores case for characters in the 7-bit ASCII range.

Synopsis

```
#include "fencode.h"
. . .
BoolT F_StrIEqualEnc(ConStringT s1,
   ConStringT s2,
   FontEncIdT feId);
```

Arguments

`s1`	A string
`s2`	A string to compare with `s1`
`feId`	The ID of the font encoding for the strings you are comparing

Returns

`True` if the strings are equal; otherwise, `False`.

Example

The following code prints the string `The strings are equal`:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
if (F_StrIEqualEnc((StringT)"string", (StringT)"String", feId))
    F_Printf(NULL, "The strings are equal.\n");
. . .
```

F_StrIEqualN()

`F_StrIEqualN()` compares two strings up to a specified number of characters, ignoring case.

IMPORTANT: This function only works with characters that use the FrameRoman encoding. This is important if your document includes Asian text.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
BoolT F_StrIEqualN(ConStringT s1,
    ConStringT s2,
    UIntT n);
```

Arguments

`s1`	A string
`s2`	A string to compare with `s1`
`n`	The number of characters to which to compare the strings

Returns

`True` if the strings are equal to `n` characters; otherwise, `False`.

Example

The following code prints the string `The first 6 chars are equal`:

```
. . .
if(F_StrIEqualN((StringT)"string1",(StringT)"String2",(IntT) 6))
   F_Printf(NULL, "The first 6 chars are equal.\n");
. . .
```

F_StrIEqualNEnc()

`F_StrIEqualNEnc()` compares two strings of the specified encoding, up to a specified number of bytes. This function ignores case for characters in the 7-bit ASCII range.

IMPORTANT: This function specifies string length in terms of bytes and not characters. Note that a string of three double-byte characters has three characters and six bytes.

Synopsis

```
#include "fencode.h"
. . .
BoolT F_StrIEqualNEnc(ConStringT s1,
   ConStringT s2,
   IntT n,
   FontEncIdT feId);
```

Arguments

`s1`	A string
`s2`	A string to compare with `s1`
`n`	The length of the strings, in bytes, by which to compare the strings
`feId`	The ID of the font encoding for the strings you are comparing

Returns

`True` if the strings are equal; otherwise, `False`.

Example

The following code determines whether to print the string `The first 12 bytes are equal`:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
StringT s1, s2;

/* Assuming you have Japanese characters in s1 and s2... */
if (F_StrIEqualNEnc(s1, s2, (IntT) 12, feId))
   F_Printf(NULL, "The first 12 bytes are equal.\n");
. . .
```

F_StrIEqualNUTF8Char ()

Compares the Unicode code points of each character in two UTF-8 strings up to a specified number of UTF-8 characters (not bytes). This function ignores cases.

Synopsis

```
#include "fencode.h"

. . .

BoolT F_StrIEqualNUTF8Char ( ConStringT s1, ConStringT s2, IntT
n);
```

Arguments

`s1`	A string
`s2`	The string to compare to `s1`

Returns

`True` if the strings are equal, `False` otherwise.

Example

The following code prints the string `First 2 chars of` Бог and Боо `are equal`

```
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
StringT s1 = F_StrCopyString("\xD0\x91\xD0\xBE\xD0\xB3");
StringT s2 = F_StrCopyString("\xD0\x91\xD0\xBE\xD0\xBE");
if(!F_StrIEqualNUTF8Char(s1, s2, 2))
   F_Printf(NULL, "First 2 chars of %s and %s are equal\n", s1,
s2);
. . .
```

F_StrIPrefixEnc()

`F_StrIPrefixEnc()` determines whether `s2` is a prefix of `s1`, checking against the specified font encoding. This function ignores case for characters in the 7-bit ASCII range.

Synopsis

```
#include "fencode.h"
. . .
BoolT F_StrIPrefixEnc(ConStringT s1,
   ConStringT s2,
   FontEncIdT feId);
```

Arguments

`s1`	The string in which to search for `s2`
`s2`	The string to search `s1 for`
`feId`	The ID of the font encoding to check against

Returns

`True` if `s2` is a prefix of `s1`, or `False` if it isn't.

Example

The following code prints the string `The second string is a prefix of the first`. Also see the example for "F_StrPrefix()" on page 694:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
StringT s1;

/* Assuming double-byte chars in s1... */
if(F_StrIPrefixEnc(s1, s1, feId)
   F_Printf(NULL, "The second string is a prefix of the
first\n");
. . .
```

F_StrIsEmpty()

`F_StrIsEmpty()` determines whether a string is empty.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
BoolT F_StrIsEmpty(StringT s);
```

Arguments

`s`	The string to check

Returns

`True` if `s` is `NULL` or the specified string is empty.

Example

The following code prints the string `s is empty` or `NULL`:

```
. . .
StringT s = NULL;
if(F_StrIsEmpty("s"))
   F_Printf(NULL, "s is empty or NULL.\n");
. . .
```

F_StrISuffixEnc()

`F_StrISuffixEnc()` determines whether `s2` is a suffix of `s1`, checking against the specified font encoding. This function ignores case for characters in the 7-bit ASCII range.

IMPORTANT: The length of the string is calculated in bytes, not characters. This is important if the string contains double-byte characters.

Synopsis

```
#include "fencode.h"
. . .
BoolT F_StrISuffixEnc(ConStringT s1,
   ConStringT s2,
   FontEncIdT feId);
```

Arguments

`s1`	The string in which to search for `s2`
`s2`	The string to search `s1` for
`feId`	The ID of the font encoding to check against

Returns

`True` if `s2` is a suffix of `s1`, or `False` if it isn't.

Example

The following code prints the string `The second string is a suffix of the first`. Also see the example for "F_StrSuffix()" on page 707:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
StringT s1;

/* Assuming double-byte chars in s1 and s2... */
if(F_StrISuffixEnc(s1, s2, feId))
   F_Printf(NULL, "The second string is a suffix of the
first\n");
. . .
```

F_StrLen()

`F_StrLen()` returns the length of a string.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrLen(ConStringT s);
```

Arguments

`s`	The string to return the length of

Returns

The length of the specified string.

Example

The following code prints the string `Length: 17`:

```
. . .
F_Printf(NULL, "Length: %d\n", F_StrLen("Athena Brighteyes"));
. . .
```

F_StrLenEnc()

`F_StrLenEnc()` returns the number of characters of a specific encoding in a string.

IMPORTANT: This function returns the length of the string in characters for the specified encoding. For some encodings, characters in the same string can be double-byte or single-byte.

Synopsis

```
#include "fencode.h"
. . .
IntT F_StrLenEnc(ConStringT s, FontEncIdT feId);
```

Arguments

`s`	The string to return the length of
`feId`	The ID of the font encoding for s

Returns

The length of the specified string.

Example

The following code prints the length of `myString`:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
StringT myString;

/* Assuming a string of ShiftJIS chars in myString... */
F_Printf(NULL, "Length: %d\n", F_StrLen(myString, feId));
. . .
```

F_StrLenUTF16()

Returns the length of a null-terminated UTF-16 string in terms of number of non-null code units (number of UChar16T).

NOTE: A UTF-16 string is considered null-terminated if it has a null UChar16T (2 bytes), rather than a null UCharT (1 byte).

Synopsis

```
#include "fencode.h"

. . .

IntT F_StrLenUTF16 (const UChar16T *string);
```

Arguments

`s`	The string whose length must be determined.

Returns

The number of non-null code units (UChar16T) in the null-terminated UTF-16 string

Example

The following code prints `The length of the string is 3 code units`

```
#include "fencode.h"
. . .
UChar16T russian_U16[]={0x0411, 0x043E, 0x0433, 0x0000};
IntT n;
F_FdeInitFontEncs((ConStringT)"UTF-8");
n = F_StrLenUTF16(russian_U16);
F_Printf(NULL, "The length of the string is %d code units", n);
...
```

F_StrListAppend()

`F_StrListAppend()` copies and appends a string to a string list. It allocates extra space if it is needed.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
ErrorT F_StrListAppend(StringListT list,
   ConStringT s);
```

Arguments

`list`	The string list
`s`	The string to append to `list`

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

The following code creates a string list and appends a single string to it:

```
. . .
StringListT list;

list = F_StrListNew((UIntT)1, (UIntT)1);
if (F_StrListAppend(list, (StringT)"Clothos"))
   F_Printf(NULL, "Couldn't allocate string list memory\n");
. . .
```

F_StrListCat()

`F_StrListCat()` concatenates two string lists.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
VoidT F_StrListCat(StringListT toList,
   StringListT appendList);
```

Arguments

`toList`	A string list
`appendList`	A string list to append to `toList`

Returns

`VoidT`.

Example

The following code creates two string lists, appending one to the other. It prints the string `Clothos Lachesis`.

```
. . .
StringListT toList, appendList;
IntT i;

toList = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(toList, (StringT)"Clothos");
appendList = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(appendList, (StringT)"Lachesis");
F_StrListCat(toList, appendList);
for(i=0; i < F_StrListLen(toList); i++)
   F_Printf(NULL, "%s ", F_StrListGet(toList, i));
. . .
```

F_StrListCopy()

`F_StrListCopy()` copies a specified string from a string list to storage you provide.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
IntT F_StrListCopy(StringListT list,
    IntT n,
    StringT s,
    IntT size);
```

Arguments

`list`	The string list containing the string to copy.
`n`	The index of the string to copy (where `0` is the index of the first string in the list).
`s`	A pointer to a block of modifiable memory for the copied string. Free this memory when you are done with it.
`size`	The number of bytes of the string to copy (including the terminating `0`).

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

The following code creates a string list and copies its first element to a string. It prints the string `Clothos has 7 chars`.

```
. . .
StringListT list;
StringT s;
IntT i;

s = F_StrNew((UIntT)8);
list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
i = F_StrListCopy(list, (IntT)0, s, (IntT)8);
F_Printf(NULL, "%s has %d chars\n", s, i);
. . .
```

F_StrListCopyList()

`F_StrListCopyList()` makes a copy of a string list.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
StringListT F_StrListCopyList(StringListT list);
```

Arguments

`list`	The string list to copy

Returns

A copy of the specified string list or `NULL` if there is insufficient memory. Use `F_StrListFree()` to free the returned string list when you are done with it.

Example

The following code creates a string list and copies it to another string list. It prints the string `Clothos`.

```
. . .
StringListT list, copy;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");

copy = F_StrListCopyList(list);
F_Printf(NULL, "%s\n", F_StrListGet(copy, 0));
. . .
```

F_StrListFirst()

`F_StrListFirst()` is a macro that returns the first string in a specified string list. It has the same effect as calling `F_StrListGet(list, 0)`.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
StringT F_StrListFirst(StringListT list);
```

Arguments

`list`	The string list from which to return the first string

Returns

The first string in the specified list.

Example

The following code creates a string list and prints its first string, `Clothos`:

```
. . .
StringListT list;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");
F_Printf(NULL, "%s\n", F_StrListFirst(list));
. . .
```

F_StrListFree()

`F_StrListFree()` frees a string list and all of the strings in it.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
VoidT F_StrListFree(StringListT list);
```

Arguments

`list`	The string list to free

Returns

`VoidT`.

Example

The following code creates a string list and frees it:

```
. . .
StringListT list;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");
F_StrListFree(list);
. . .
```

F_StrListGet()

`F_StrListGet()` returns a specified string in a string list. If the string list is freed or changed, the string returned by `F_StrListGet()` is invalidated. `F_StrListGet()` provides fast access to the strings in the string list.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
StringT F_StrListGet(StringListT list,
    IntT n);
```

Arguments

`list`	The string list
`n`	The index of the string to get (where `0` is the index of the first string in the list)

Returns

The specified string, or `NULL` if it couldn't find the string. If you change or free the returned string, it has side effects on the string list.

Example

The following code creates a string list and prints all of its strings:

```
. . .
StringListT list;
IntT i;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");
for(i=0; i < F_StrListLen(list); i++)
    F_Printf(NULL, "%s ", F_StrListGet(list, i));
. . .
```

F_StrListIIndex()

`F_StrListIIndex()` returns the position of the first occurrence (ignoring case) of a string in a specified string list.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
IntT F_StrListIIndex(StringListT list,
   StringT s);
```

Arguments

`list`	The string list
`s`	The string to find the first occurrence of

Returns

The position of `s` in the list, or `-1` if `s` does not occur in list.

Example

The following code creates a string list and finds the string `clothos` in it (ignoring case). It prints the string `clothos at 0th position`.

```
. . .
StringListT list;
IntT i;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");
i = F_StrListIIndex(list, "clothos");
if(i != -1)
   F_Printf(NULL, "clothos at %dth position\n", i);
. . .
```

F_StrListIndex()

`F_StrListIndex()` returns the position of the first occurrence of a string in a specified string list.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
IntT F_StrListIndex(StringListT list,
   StringT s);
```

Arguments

`list`	The string list
`s`	The string to find the first occurrence of

Returns

The position of `s` in the list, or `-1` if `s` does not occur in list.

Example

The following code creates a string list and finds the string `clothos` in it. It prints the string `Lachesis at 1th position`.

```
. . .
StringListT list;
IntT i;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");
i = F_StrListIndex(list, "Lachesis");
if(i != -1)
   F_Printf(NULL, "Lachesis at %dth position\n", i);
. . .
```

F_StrListInsert()

`F_StrListInsert()` inserts a string into a list at a specified location. It allocates extra space if necessary.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
ErrorT F_StrListInsert(StringListT list,
   StringT s,
   UIntT position);
```

Arguments

`list`	The string list
`s`	The string to insert
`position`	The position at which to insert the string in the list (where `0` is the position of the first string in the list)

Returns

`FdeSuccess` if it succeeds, or a nonzero value if it fails.

Example

The following code creates a string list with two elements and inserts the string `atropos` between them.

```
. . .
StringListT list;
IntT i;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");

F_StrListInsert(list, "Atropos", 1);
. . .
```

F_StrListLast()

`F_StrListLast()` is a macro that returns the last string in a specified string list. It has the same effect as calling `F_StrListGet(list, F_StrListLen(list)-1)`.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
StringT F_StrListLast(StringListT list);
```

Arguments

`list`	The string list

Returns

The last string in the specified list.

Example

The following code creates a string list and prints its last string, `Lachesis`:

```
. . .
StringListT list;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");
F_Printf(NULL, "%s\n", F_StrListLast(list));
. . .
```

F_StrListLen()

`F_StrListLen()` returns the number of strings in a specified list.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
IntT F_StrListLen(StringListT list);
```

Arguments

`list`	The string list

Returns

The number of strings in the list.

Example

The following code creates a string list and prints its length, `Length 2`:

```
. . .
StringListT list;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");
F_Printf(NULL, "Length: %d", F_StrListLen(list));
. . .
```

F_StrListNew()

`F_StrListNew()` allocates a new string list with a specified number of strings.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
StringListT F_StrListNew(UIntT numStrings,
   UIntT quantum);
```

Arguments

`numStrings`	The number of strings in the new list
`quantum`	The allocation quantum for increasing the size of the list

Returns

The created string list, or `NULL` if the list can't be allocated. When you are finished with the returned string list, use `F_StrListFree()` to free it.

Example

The following code creates a string list, adds a string to it, and then frees the string list:

```
. . .
StringListT list;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListFree(list);
. . .
```

F_StrListRemove()

`F_StrListRemove()` deletes a specified string from a string list.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
VoidT F_StrListRemove(StringListT list,
   UIntT position);
```

Arguments

`list`	The string list
`position`	The position of the string to remove (where `0` is the position of the first string in the list)

Returns

`VoidT`.

Example

The following code creates a string list and removes the first string from it. It prints the string that becomes the first string, `Lachesis`.

```
. . .
StringListT list;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");
F_StrListRemove(list, (UIntT)0);
F_Printf(NULL, "%s", F_StrListFirst(list));
. . .
```

F_StrListSetString()

`F_StrListSetString()` copies and sets a specified string at a specified location in a string list.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
ErrorT F_StrListSetString(StringListT list,
    ConStringT s,
    UIntT position);
```

Arguments

`list`	The string list
`s`	The string to set in the string list
`position`	The position at which to set the string specified by `s` (where `0` is the position of the first string in the list)

Returns

`FdeSuccess` if it succeeds, or a nonzero value if there isn't enough memory or the specified position is out of bounds.

Example

The following code creates a string list and sets the second string in it. It prints the string `Clothos Atropos`.

```
. . .
StringListT list;
IntT i;

list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");
F_StrListSetString(list, (StringT)"Atropos", (UIntT)1);
for(i=0; i < F_StrListLen(list); i++)
    F_Printf(NULL, "%s ", F_StrListGet(list, i));
. . .
```

F_StrListSort()

`F_StrListSort()` performs a destructive sort of a string list using a quick sort algorithm with a specified comparison function.

Synopsis

```
#include "fdetypes.h"
#include "fstrlist.h"
. . .
VoidT F_StrListSort(StringListT list,
   NativeIntT (*fn) (ConStringT *, ConStringT *));
```

Arguments

`list`	The string list
`fn`	The comparison function used to sort the string list

The comparison function should have the following prototype:

```
NativeIntT fn(ConStringT *s1,
   ConStringT *s2);
```

The comparison function should return `0` if `s1` and `s2` are equal, a negative value when `s1` is less than `s2`, and a positive value when `s1` is greater than `s2`. `F_StrListSort()` sorts the elements of the specified list in ascending order.

Returns

`VoidT`.

Example

The following code creates a string list and sorts it. It prints the string `Atropos Clothos Lachesis`.

```
. . .
NativeIntT fn(ConSringT *s1, ConSringT *s2)
{
   return (F_StrCmp(*s1, *s2));
}
. . .
StringListT list;
IntT i;

list = F_StrListNew((UIntT)3, (UIntT)1);
F_StrListAppend(list, (StringT)"Clothos");
F_StrListAppend(list, (StringT)"Lachesis");
F_StrListAppend(list, (StringT)"Atropos");
F_StrListSort(list, &fn);
for(i=0; i < F_StrListLen(list); i++)
   F_Printf(NULL, "%s ", F_StrListGet(list, i));
. . .
```

F_StrNew()

`F_StrNew()` allocates a string using `F_Calloc()`.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
StringT F_StrNew(UIntT len);
```

Arguments

`len`	The length of the string to create

Returns

The allocated string, or `NULL` if it fails. When you are finished with the returned string, use `F_Free()` to free it. For more information, see “F_Free()” on page 559.

Example

The following code allocates memory for a 19-character string to `s`, and then frees the memory:

```
. . .
StringT s;
s = F_StrNew((IntT)20);
if(s != NULL) F_Free(s);
. . .
```

See also

"F_StrCopyString()" on page 658, "F_StrCpy()" on page 659, and "F_Free()" on page 559.

F_StrPrefix()

`F_StrPrefix()` determines whether `s2` is a prefix of `s1`.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
BoolT F_StrPrefix(ConStringT s1,
   ConStringT s2);
```

Arguments

`s1`	The string in which to search for `s2`
`s2`	The string to search `s1` for

Returns

`True` if `s2` is a prefix of `s1`, or `False` if it isn't.

Example

The following code prints the string `The second string is a prefix of the first`:

```
. . .
if(F_StrPrefix((StringT)"an elk", (StringT)"an e"))
   F_Printf(NULL,
      "The second string is a prefix of the first\n");
. . .
```

F_StrPrefixN()

`F_StrPrefixN` determines whether a specified portion of a string is a prefix of another string.

IMPORTANT: The length to compare is calculated in bytes, not characters. This is important if the string contains double-byte characters.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
BoolT F_StrPrefixN(ConStringT s1,
   ConStringT s2,
   UIntT n);
```

Arguments

`s1`	A string
`s2`	A string containing the characters to test as a prefix of `s1`
`n`	The number of bytes in `s2` to test as a prefix of `s1`

Returns

`True` if `n` characters of `s2` are a prefix of `s1`, or `False` if they aren't.

Example

The following code prints the string `The second string is a prefix of the first`:

```
. . .
if(F_StrPrefixN((StringT)"an elk",(StringT)"anne elk",(IntT) 2))
   F_Printf(NULL,
      "The second string is a prefix of the first\n");
. . .
```

F_StrRChr()

`F_StrRChr()` returns a pointer to the first occurrence of a character in a string, starting from the end of the string.

IMPORTANT: This function only works with characters that use the FrameRoman encoding. This is important if your document includes Asian text.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
StringT F_StrRChr(StringT s,
    PUCharT c);
```

Arguments

`s`	The string to search for `c`
`c`	The character to search for in `s`

Returns

A pointer to the first occurrence of `c` in `s`, starting from the end of the string, or `NULL` if `c` does not occur in `s`.

Example

The following code prints the string `otafar`:

```
. . .
StringT s, string;
s = F_StrCopyString("Apollo Shootafar");
string = F_StrRChr(s, 'o');
F_Printf(NULL, "%s\n", string);
. . .
```

F_StrRChrEnc()

`F_StrRChrEnc()` returns a pointer to the first occurrence in a string, starting from the end of the string, of a double-byte character of the specified encoding.

Synopsis

```
#include "fencode.h"
. . .
StringT F_StrRChrEnc(ConStringT s,
    UCharT first,
    UCharT second,
    FontEncIdT feId);
```

Arguments

`s`	The string to search
`first`	The first byte of the character to search `s` for
`second`	The second byte of the character to search `s` for
`feId`	The ID of the font encoding to check against

Returns

A pointer to the last occurrence in `s` of the double byte character represented by `first` and `second`. Returns `NULL` if such a character does not occur in `s`.

Example

The following code searches from the end of `s` and returns a string beginning with the Japanese character `89C3`. Also see the example for "F_StrRChr()" on page 695:

```
. . .
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
StringT s, string;

/* Assuming a string of double-byte chars in s... */
string = F_StrRChrEnc(s, \x89, \xC3, feId);
. . .
```

F_StrReverse()

`F_StrReverse()` reverses a specified number of characters of a string.

IMPORTANT: This function only works with characters that use the FrameRoman encoding. This is important if your document includes Asian text.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
VoidT F_StrReverse(StringT s,
    IntT l);
```

Arguments

`s`	The string to reverse.
`l`	The number of characters to reverse. If `l` is greater than the length of the string, `F_StrReverse()` reverses all the characters in the string.

Returns

`VoidT`.

Example

The following code prints the string `ablE was I ere I saw elbA`:

```
. . .
StringT s;
s = F_StrCopyString("Able was I ere I saw Elba");
F_StrReverse(s, 100);
F_Printf(NULL, "%s\n", s);
. . .
```

F_StrReverseUTF8Char ()

Reverses a specified number of characters in a UTF-8 string.

IMPORTANT: This reverses the string code-point-by-code-point (rather than byte-by-byte). However, this still results in absurdities like the diaeresis coming before 'a'when

it was after 'a' in the original string. Thus äo will become öa if ä has been stored as 'a' followed by the diaresis.

Synopsis

```
#include "fencode.h"

. . .

VoidT F_StrReverseUTF8Char (StringT s, IntT l);
```

Arguments

`s`	The string to reverse.
`l`	The number of characters to reverse. If `l` is greater than the length of the string, `F_StrReverse()` reverses all the characters in the string.

Returns

`VoidT`

Example

The following code prints the string २३४ + ३२४ = 558

```
StringT devanagiri_four="\xE0\xA5\xAA";
StringT devanagiri_three="\xE0\xA5\xA9";
StringT devanagiri_two ="\xE0\xA5\xA8";
UCharT n_234[256];
UCharT n_324[256];
IntT res;

FontEncIdT feId = F_FdeInitFontEncs((ConStringT)"UTF-8");
F_StrTruncEnc(n_234,0,feId);
F_StrTruncEnc(n_324,0,feId);

F_StrCpy(n_234, devanagiri_two);
F_StrCat(n_234, devanagiri_three);
F_StrCat(n_234, devanagiri_four);

F_StrCpy(n_324,n_234);
F_StrReverseUTF8Char(n_324,2);

res = F_StrAlphaToIntUnicode(n_234) +
F_StrAlphaToIntUnicode(n_324);
F_Printf(NULL,"%s + %s = %d", n_234, n_324, res);
...
```

F_StrStrip()

`F_StrStrip()` removes a specified set of characters from a string.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
VoidT F_StrStrip(StringT s,
   ConStringT strip);
```

Arguments

`s`	The string from which to remove characters
`strip`	The characters to remove from `s`

Returns

`VoidT`.

Example

The following code prints the string `Wrttn Hbrw mts shrt vwls`:

```
. . .
StringT s;
s = F_StrCopyString("Written Hebrew omits short vowels.");
F_StrStrip(s, (StringT) "aeiouy");
F_Printf(NULL, "%s\n", s);
. . .
```

F_StrStripLeadingSpaces()

`F_StrStripLeadingSpaces()` removes the leading spaces in a string.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
VoidT F_StrStripLeadingSpaces(StringT s);
```

Arguments

`s`	The string from which to remove leading spaces

Returns

`VoidT`.

Example

The following code prints the string `leading spaces`:

```
. . .
StringT s;
s = F_StrCopyString("    leading spaces");
F_StrStripLeadingSpaces(s);
F_Printf(NULL, "%s\n", s);
. . .
```

F_StrStripTrailingSpaces()

`F_StrStripTrailingSpaces()` removes the trailing spaces in a string.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
VoidT F_StrStripTrailingSpaces(StringT s);
```

Arguments

`s`	The string from which to remove trailing spaces

Returns

`VoidT`.

Example

The following code prints the string `trailing spaces`:

```
. . .
StringT s;
s = F_StrCopyString("trailing spaces   ");
F_StrStripTrailingSpaces(s);
F_Printf(NULL, "%s\n", s);
. . .
```

F_StrStripUTF8Chars ()

Removes a specified set of UTF-8 characters from a UTF-8 string

IMPORTANT: This function doesn't perform canonical or compatibility normalizations. Because it works on a code-point-by-code-point basis, it treats 'a' followed by 'diaeresis' as separate entities despite their forming the grapheme cluster ä.

If "äo" must be stripped from "göat" and ä (and ö) are stored as 'a' (and 'o') followed by 'diaeresis' , the result is "gt" because 'a', 'diaeresis' and 'o' are all removed from the string. If, however, ö is stored in its composed form, the result is "göt". If both ö and ä are stored in their composed form, the result is "göat".

Synopsis

```
#include "fencode.h"

. . .

VoidT F_StrStripUTF8Chars (StringT s, ConStringT strip);
```

Arguments

`s`	The string from which to remove characters
`strip`	The characters to remove from `s`

Returns

`VoidT`

Example

The following code prints the string `Have you` göt a göat?

```
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
StringT orig = F_StrCopyString("g\xC3\xB6\xC1\x74");
StringT strip = "a\xCC\x88\x6F"; /* CC 88 is diaeresis */
UCharT cop[256];
cop[0]=0;
F_StrCpy(cop,orig);
F_StrStripUTF8Chars(cop, strip);
F_Printf(NULL, "Have you %s a %s?\n", cop, orig);
. . .
```

F_StrStripUTF8String ()

Removes all occurrences of one string from another

IMPORTANT: Because this function doesn't perform canonical or compatibility normalizations, it treats 'a' followed by 'diaeresis' as different from the precomposed form ä (code point 0x00E4).

Removing "hel" from "hehello" results in "helo" and not "o". That is, this performs only one pass.

Synopsis

```
#include "fencode.h"

. . .

VoidT F_StrStripUTF8String (StringT s, ConStringT strip);
```

Arguments

`s`	A string
`strip`	The string whose occurrence must be removed from `s`

Returns

`VoidT`

Example

The following code prints the string `helo`

```
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
StringT orig = F_StrCopyString("hehello");
StringT strip = "hel";
F_StrStripUTF8Chars(orig, strip);
F_Printf(NULL, orig);
. . .
```

F_StrStripUTF8Strings ()

Removes all occurrences of one set of strings from another

IMPORTANT: Because this function doesn't perform canonical or compatibility normalizations, it treats 'a' followed by 'diaeresis' as different from the precomposed form ä (code point 0x00E4).

Synopsis

```
#include "fencode.h"

. . .

VoidT F_StrStripUTF8Strings (StringT s, StringListT sstrip);
```

Arguments

`s`	A string
`sstrip`	The list of strings whose occurrences must be removed from `s`

Returns

`VoidT`

Example

The following code prints the string Vision is a daydream. Action is a nightmare.

```
. . .
F_FdeInitFontEncs((ConStringT)"UTF-8");
StringT orig = F_StrCopyString("Vision without action is a
daydream.
Action without vision is a nightmare.");
StringListT list;
list = F_StrListNew((UIntT)1, (UIntT)1);
F_StrListAppend(list, (StringT)"action ");
F_StrListAppend(list, (StringT)"without ");
F_StrListAppend(list, (StringT)"vision ");

F_StrStripUTF8Strings(orig, list);
F_Printf(NULL, orig);
. . .
```

F_StrSubString()

`F_StrSubString()` finds the first occurrence of a specified string in another string.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
IntT F_StrSubString(ConStringT s1,
   ConStringT s2);
```

Arguments

`s1`	The string in which to search for `s2`
`s2`	The string to search for

Returns

The index of the first occurrence of `s2` in `s1`, or `-1` if `s2` doesn't occur in `s1`.

Example

The following code prints the string `Earth found at the 9th char`:

```
. . .
StringT s1;
IntT i;
s1 = F_StrCopyString("Poseidon Earthshaker");
i = F_StrSubString(s1, (StringT) "Earth");
if(i != -1) F_Printf(NULL, "Earth found at the %dth char\n", i);
. . .
```

F_StrSuffix()

`F_StrSuffix()` determines whether a specified string is a suffix of another string.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
BoolT F_StrSuffix(ConStringT s1,
   ConStringT s2);
```

Arguments

`s1`	The string to test for the suffix `s2`
`s2`	The string to test as a suffix of `s1`

Returns

`True` if `s2` is a suffix of `s1`, or `False` if it isn't.

Example

The following code prints the string `The second string is a suffix of the first`:

```
. . .
if(F_StrSuffix((StringT)"an elk", (StringT)"elk"))
   F_Printf(NULL,
      "The second string is a suffix of the first\n");
. . .
```

F_StrTok()

`F_StrTok()` is equivalent to `strtok()`. It returns the text tokens in a specified string, separated by occurrences of any combination of one or more of the characters in another string.

IMPORTANT: This function only works with characters that use the FrameRoman encoding. This is important if your document includes Asian text.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
StringT F_StrTok(StringT s1,
    ConStringT s2);
```

Arguments

`s1`	The string to parse for text tokens
`s2`	A string containing characters used to separate the text tokens

Returns

The first time you call `F_StrTok()`, it returns a pointer to the beginning of the first token in `s1` and overwrites `s1` with `NULL` at the end of the token. `F_StrTok()` keeps track of its position in the string after each call so that when you call it again with `s1` set to `NULL`, it starts immediately after the previous token. The separator string `s2` can be different for each call. When it has passed all the tokens in `s1`, `F_StrTok()` returns `NULL`.

Example

The following code prints the string `The total is 14`:

```
. . .
StringT s, s1;
IntT i = 0;
s1 = F_StrCopyString((StringT)"4 plus 7 p 3");
s = F_StrTok(s1, (StringT)" plus ");
while (s != NULL)
{
    i += F_StrAlphaToInt(s);
    s = F_StrTok(NULL, (StringT)" plus ");
}
F_Printf(NULL, "The total is %d\n", i);
. . .
```

F_StrTokUTF8 ()

Is similar to `F_StrTok` except that it considers UTF-8 characters and not only single bytes. It returns the text tokens in a specified UTF-8 string, separated by occurrences of any combination of one or more of the UTF-8 characters in another string.

Synopsis

```
#include "fencode.h"

. . .

StringT F_StrTokUTF8 (StringT s1, ConStringT s2);
```

Arguments

`s1`	The string to parse for text tokens
`s2`	A string containing characters used to separate the text tokens

Returns

The first time you call `F_StrTokUTF8()`, it returns a pointer to the beginning of the first token in `s1` and overwrites `s1` with `NULL` at the end of the token. `F_StrTokUTF8()` keeps track of its position in the string after each call so that when you call it again with `s1` set to `NULL`, it starts immediately after the previous token. The separator string `s2` can be different for each call. When it has passed all the tokens in `s1`, `F_StrTokUTF8()` returns `NULL`.

Example

The following code prints the string `4 plus ३ p 3 + 10 ⊕ 23 = 42`

```
. . .
StringT s, s1,cop, tokens;
IntT i = 0;
F_FdeInitFontEncs("UTF-8");

s1 = F_StrCopyString("4 plus \xE0\xA5\xA8 p 3 + 10 \xE2\x8A\x95
23");
cop = F_StrCopyString(s1);

tokens = " +plus\xE2\x8A\x95";
s = F_StrTokUTF8(s1, tokens);

while (s != NULL)
    {
    i += F_StrAlphaToIntUnicode(s);
    s = F_StrTok(NULL, tokens);
    }
F_Printf(NULL, "%s = %d",cop,i);
...
```

F_StrTrunc()

`F_StrTrunc()` truncates a string at a specified position. It overwrites the character at the specified position with `'\0'`.

IMPORTANT: This function only works with characters that use the `FrameRoman` encoding. If your document includes Asian text, use `F_StrTruncEnc()`.

Synopsis

```
#include "fdetypes.h"
#include "fstrings.h"
. . .
VoidT StringT F_StrTrunc(StringT s,
    UIntT n);
```

Arguments

`s`	The string to truncate
`n`	The position at which to truncate `s`

IMPORTANT: If the value for `n` is greater than the length of the string, you will overwrite an unspecified location in memory with a NULL byte.

Returns

`VoidT`.

Example

The following code prints the string `wine`:

```
. . .
StringT s;
s = F_StrCopyString("wine-dark sea");
F_StrTrunc(s, 4);
F_Printf(NULL, "%s\n", s);
. . .
```

F_StrTruncEnc()

`F_StrTruncEnc()` truncates a string at a specified position. It overwrites the character at the specified position with `'\0'`.

IMPORTANT: The position is specified in terms of bytes and not characters. In a string of three double-byte characters, to truncate after the second character you would specify a value of 4 for `n`.

Synopsis

```
#include "fencode.h"
. . .
VoidT F_StrTruncEnc(StringT s,
   UIntT n,
   FontEncIdT feId);
```

Arguments

`s`	The string to truncate
`n`	The position at which to truncate `s`
`feId`	The ID of the font in string `s`

IMPORTANT: If the value for `n` is greater than the length of the string, you will overwrite an unspecified location in memory with a NULL byte.

Returns

`VoidT`.

Example

The following code prints the characters in the first 12 bytes of a double-byte string. Also see the example for "F_StrTrunc()" on page 710:

```
. . .
StringT s;
FontEncIdT feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");

. . .
/* Assuming s is a string of double-byte characters... */
F_StrTruncEnc(s, 12, feId);
F_Printf(NULL, "%s\n", s);
. . .
```

F_TextEncToFontEnc()

Converts from `FTextEncodingT` to `FontEncIdT`

NOTE: This function has different behaviors for *Compatibility Mode* and *Unicode Mode*. In *Unicode Mode*, if no match is found, UTF-8 is the default output. In *Compatibility Mode*, the default output is FrameRoman.

Synopsis

```
#include "fencode.h"

. . .

FontEncIdT F_TextEncToFontEnc (FTextEncodingT textEnc);
```

Arguments

`textEnc`	The text encoding to convert

Returns

The font encoding corresponding to the text encoding

Example

The following code prints "JISX0208.ShiftJIS"

```
#include "fencode.h"
. . .
StringT fname =
F_FontEncName(F_TextEncToFontEnc(F_EncShiftJIS));
F_FdeInitFontEncs((ConStringT) "UTF-8");
F_Printf(NULL, fname);
...
```

F_UnlockHandle()

`F_UnlockHandle()` unlocks a specified handle and decreases the lock count. The handle remains locked if the lock count is greater than zero. After you unlock the handle, the memory associated with it can be relocated again if necessary. You can also use `F_FreeHandle()` and `F_ReallocHandle()` on the handle.

Synopsis

```
#include "fdetypes.h"
#include "fmemory.h"
. . .
IntT F_UnlockHandle(HandleT handle);
```

Arguments

`handle`	The handle to unlock

Returns

The lock count of the handle after `F_UnlockHandle()` unlocks it.

Example

The following code unlocks a handle:

```
. . .
HandleT hndl = NULL;

if(F_UnlockHandle(hndl) != 0)
    F_Printf(NULL, "Handle is still locked.\n");
. . .
```

See also

"F_AllocHandle()" on page 69 and "F_LockHandle()" on page 577.

F_UTF16CharSize()

Returns the number of UTF-16 code units taken by the UTF-16 character based on the first code unit of the character.

UTF-16 encoding allows you to determine the number of code units a character occupies by looking at the first code unit of the character. A UTF-16 code unit is 2 bytes. A Unicode character in UTF-16 occupies 1 to 2 UTF-16 code units.

Synopsis

```
#include "fencode.h"

. . .

IntT F_UTF16CharSize (const UChar16T c);
```

Arguments

c	The first code unit of the character whose size must be found

Returns

The number of code units occupied by the UTF-16 character (1-2)

Example

The following code prints ɣ occupies 1 UTF-16 code units

```
#include "fencode.h"
. . .
StringT devanagiri_four="\xE0\xA5\xAA";
UChar16T devanagiri_four_U16[2];
IntT n;

F_FdeInitFontEncs((ConStringT)"UTF-8");
F_CharUTF8ToUTF16(devanagiri_four, devanagiri_four_U16);
n = F_UTF16CharSize(*devanagiri_four_U16);
F_Printf("%C occupies %d UTF-16 code units", devanagiri_four,
n);
...
```

F_UTF16NextChar()

Increments the UTF-16 character pointer to the next character in the string.

You must use this function to traverse a UTF-16 string. Using cp++ to parse a string, where cp is a character pointer (UChar16T pointer), is incorrect for UTF-16 because it traverses the string on a code-unit-by-code-unit basis, rather than a character-by-character basis. This function also returns the incremented pointer.

Synopsis

```
#include "fencode.h"

. . .

const UChar16T * F_UTF16NextChar (const UChar16T **c);
```

Arguments

c	Pointer to the character pointer that must be incremented

Returns

The incremented character pointer.

Example

The following code prints Skipping 2 characters we get r

```
#include "fencode.h"
. . .
UCharT russian_char_U8[4];
UChar16T russian_U16[]={0x0411,0x043E,0x0433,0x0000};
UChar16T *t=russian_U16;

F_FdeInitFontEncs((ConStringT)"UTF-8");
F_UTF16NextChar(&t);
F_UTF16NextChar(&t);
F_CharUTF16ToUTF8(t,russian_char_U8);

F_Printf(NULL, "Skipping 2 characters we get %C",
russian_char_U8);
...
```

F_UTF8CharSize()

Returns the number of bytes taken by the UTF-8 character based on its first byte.

UTF-8 encoding allows you to determine the number of bytes a character occupies by lookingat the first byte of the character. A Unicode character in UTF-8 occupies 1 to 4 bytes .

Synopsis

```
#include "fencode.h"

. . .

IntT F_UTF8CharSize (const UCharT c);
```

Arguments

c	The first byte of the character whose size must be found

Returns

The number of bytes occupied by the UTF-8 character (1-4)

Example

The following code prints ɣ `occupies 3 bytes`

```
#include "fencode.h"
. . .
StringT devanagiri_four="\xE0\xA5\xAA";
IntT n;
F_FdeInitFontEncs((ConStringT)"UTF-8");
n = F_UTF8CharSize(*devanagiri_four);
F_Printf("%C occupies %d bytes", devanagiri_four, n);
. . .
```

F_UTF8NextChar()

Increments the UTF-8 character pointer to the next character in the string.

You must use this function to traverse a UTF-8 string. Using cp++ to parse a string, where cp is a character pointer, is incorrect for UTF-8 because it traverses the string on a byte-by-byte basis, rather than a character-by-character basis. This function also returns the incremented pointer.

Synopsis

```
#include "fencode.h"

. . .

const UCharT * F_UTF8NextChar (const UCharT **c);
```

Arguments

`c`	Pointer to the character pointer that must be incremented

Returns

The incremented character pointer.

Example

The following code prints Бог-Б-о-г

```
#include "fencode.h"
. . .
StringT russian="\xD0\x91\xD0\xBE\xD0\xB3";
UCharT *t=russian;
F_FdeInitFontEncs((ConStringT)"UTF-8");
F_Printf(NULL, russian);
while(*t)
{
    F_Printf(NULL,"-%C",t);
    F_UTF8NextChar(&t);
}
...
```

F_Warning()

`F_Warning()` reports a specified warning message to the Frame console.

Synopsis

```
#include "fdetypes.h"
#include "fprogs.h"
. . .
VoidT F_Warning(StringT msg);
```

Arguments

`msg`	The warning message to display

Returns

`VoidT`.

Example

The following code prints a warning:

```
. . .
F_Warning("A Warning.\n");
. . .
```

F_WriteBytes()

`F_WriteBytes()` writes a specified number of bytes from a pointer to a channel. It swaps bytes if you have specified a byte order with `F_SetByteOrder()` or `F_ResetByteOrder()` and the byte order is different from the current platform's byte order.

Synopsis

```
#include "fdetypes.h"
#include "fioutils.h"
. . .
IntT F_WriteBytes(PtrT ptr,
    IntT numBytes,
    ChannelT channel);
```

Arguments

`ptr`	A pointer to the bytes to write
`numBytes`	The number of bytes to write
`channel`	The channel to which to write the bytes

Returns

The number of bytes actually written.

Example

The following code writes the string `Every good boy deserves fudge` to a file located in the default directory:

```
. . .
static UCharT buf[] = "Every good boy deserves fudge";
ChannelT chan;
FilePathT *path;

path = F_PathNameToFilePath((StringT)"test.txt",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"w")) == NULL) return;
F_WriteBytes((PtrT)buf, F_StrLen(buf), chan);
. . .
```

F_WriteLongs()

`F_WriteLongs()` writes a specified number of longs (4 bytes) from a pointer to a channel. It swaps bytes if you have specified a byte order with `F_SetByteOrder()` or `F_ResetByteOrder()` and the byte order is different from the current platform's byte order.

Synopsis

```
#include "fdetypes.h"
#include "fioutils.h"
. . .
IntT F_WriteLongs(PtrT ptr,
    IntT num,
    ChannelT channel);
```

Arguments

`ptr`	A pointer to the longs to write
`num`	The number of longs to write
`channel`	The channel to which to write the longs

Returns

The number of longs actually written.

Example

The following code writes the long integers 65535, 5678, and 90123 to a file located in the default directory:

```
. . .
IntT buf[] = {65535, 5678, 90123};
ChannelT chan;
FilePathT *path;

path = F_PathNameToFilePath((StringT)"test.dat",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"w")) == NULL) return;
F_WriteLongs((PtrT)buf, 3, chan);
. . .
```

F_WriteShorts()

`F_WriteShorts()` writes a specified number of shorts (2 bytes) from a pointer to a channel. It swaps bytes if you have specified a byte order with `F_SetByteOrder()` or `F_ResetByteOrder()` and the byte order is different from the current platform's byte order.

Synopsis

```
#include "fdetypes.h"
#include "fioutils.h"
. . .
IntT F_WriteShorts(PtrT ptr,
   IntT n,
   ChannelT channel);
```

Arguments

`ptr`	A pointer to the shorts to write
`n`	The number of shorts to write
`channel`	The channel to which to write the shorts

Returns

The number of shorts actually written.

Example

The following code writes the short integers 6, 34, and –87 to a file in the default directory:

```
. . .
ShortT buf[] = {6, 34, -87};
ChannelT chan;
FilePathT *path;

path = F_PathNameToFilePath((StringT)"test.dat",
                NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"w")) == NULL) return;
F_WriteShorts((PtrT)buf, 3, chan);
. . .
```

4 Object Reference

This chapter lists API objects and their properties. For a detailed description of how the API represents things in FrameMaker products, see Part II, "Frame Product Architecture," in the *FDK Programmer's Guide*.

Read-only properties are indicated with a dagger (†). All other properties are read/write.

If there is a specific range of valid values for a property, it is indicated in parentheses in the Meaning column. If a property must specify an ID for a particular type of object, the object type is indicated in parentheses.

Books

The API uses `FO_Book` objects to represent books and `FO_BookComponent` objects to represent book components.

FO_Book

An `FO_Book` object represents a FrameMaker product book. `FO_Book` objects have the following properties.

Book general properties

The following `FO_Book` properties apply to all FrameMaker product books.

Property	Type	Meaning
`FP_BookDontUpdateReferences`	`IntT`	`False` if the FrameMaker product updates cross-references when it opens the book.
`FP_BookIsModified`†	`IntT`	`True` if the book has been modified.
`FP_BookIsSelected`	`IntT`	`True` if the book icon in the book window is selected.

Property	Type	Meaning
FP_FirstComponentInBook	F_ObjHandleT	First component in the book (FO_BookComponent ID).
FP_FirstSelectedComponentInBook	F_ObjHandleT	First selected component in the book (FO_BookComponent ID).
FP_IsIconified	IntT	True if the book window is iconified .
FP_IsInFront	IntT	True if the book window is in front of other windows in the FrameMaker product session.
FP_IsOnScreen	IntT	True if the book is visible on the screen. Note that this property is always True for books, and setting it to False has no effect.
FP_Label	StringT	The title in the book window title bar.
FP_Name	StringT	Pathname of the book.
FP_NextOpenBookInSession†	F_ObjHandleT	Next open book in session's list of open books (FO_Book ID).
FP_StatusLine	StringT	String that appears in the book status bar. In versions before 6.0, this property always returned an empty string; In versions 6.0 and later, it returns the current status string. If you set string content to FP_StatusLine (a string other than ""), the string will remain in the status bar until you reset it. To reset FP_StatusLine so the FrameMaker product automatically updates the status line with normal status information, set it to an empty string ("").
FP_ScreenHeight	IntT	Height of the book window in pixels.

Property	Type	Meaning
FP_ScreenWidth	IntT	Width of the book window in pixels.
FP_ScreenX	IntT	The offset of the book window in pixels from the left side of the screen (or the left of the FrameMaker product application window on Windows). If you set a value that would result in the book window being off the screen, that value is ignored and the old value is retained.
FP_ScreenY	IntT	The offset of the book window in pixels from the top of the screen (or the top of the FrameMaker product application window on Windows). If you set a value that would result in the book window being off the screen, that value is ignored and the old value is retained.
FP_ShowElementDescriptiveNames	IntT	If true, then show element descriptive names in element catalog, as specified in element definition.
FP_TypeOfDisplayText	IntT	The type of text snippet to display for each icon in the book window: FV_BK_FILENAME displays the book component's filename FV_BK_TEXT displays the first paragraph of the component's first flow
FP_UseInitialStructureOfAutoInsertedElements	IntT	If true, then auto-insertion rules will be processed recursively. For example, if an element is inserted automatically, and auto-insertion rules exist for this element in the element definition, then those rules will also be processed.

Property	Type	Meaning
FP_XmlDocType	StringT	The DOCTYPE parameter rom the source XML.
FP_XmlEncoding	StringT	The encoding parameter of the XML Declaration for the source XML. The string is empty if no encoding is specified. If this property is set, the XML Declaration will contain the encoding parameter with this value on Save As XML. For information about the encoding that FrameMaker supports with a default installation, see the online manual, the *Structure Application Developer's Guide.*
FP_XmlFileEncoding	StringT	The encoding that was detected for the source XML book. If no encoding was specified for the source XML, FP_XmlEncoding will be an empty string. In that case, if this stringis set it will determine the encoding to use when saving as XML. If P_XmlEncoding has a value, this string may be empty. For information about the encoding that FrameMaker supports with a default installation, see the online manual, the *Structure Application Developer's Guide*.
FP_XmlPublicId	StringT	The DOCTYPE public identifier for the source XML document

Property	Type	Meaning
FP_XmlStandAlone	IntT	An integer that specifies the XML standalone parameter for the XML document that was the source of this document. Can be one of • FV_XML_STANDALONE_YES • FV_XML_STANDALONE_NO • FV_XML_STANDALONE_NA The standalone parameter is declared in the XML Declaration. For a file with no XML Declaration, the value is FV_XML_STANDALONE_NODEC. For an XML Declaration with no standalone parameter, this value is FV_XML_STANDALONE_NONE.
FP_XmlStyleSheet	StringT	The XML stylesheet processing instruction to write out to XML when saving the book as XML. The FDK does not verify that you use correct syntax in this string. The string you set should not include the PI delimiters, <? and ?>. For example, the string you supply for my.css would be: "type=\"text\\css\" href=\"my.css\"" Only use this string to set a specific stylesheet specification. F_ApiGetString() always returns NULL for this parameter. To get the list of stylesheet specifications associated with a book, use FP_XmlStyleSheetList, below.

Property	Type	Meaning
FP_XmlStyleSheetList	F_StringsT	A list of stylesheet processing instructions for the current document. One document can have more than one stylesheet specification associated with it. The FDK does not verify that you use correct syntax in these strings. The strings should not include the PI delimiters, <? and ?>. For example, the string you supply for my.css would be: "type=\"text\\css\"href=\"my.css\"" Setting a list to FP_XmlStyleSheetList completely overwrites the preceeding list.
FP_XmlSystemId	StringT	

Property	Type	Meaning
FP_XmlUseBOM	IntT	Indicates whether a byte order mark was detected when opening the source XML. Can be one of: • FV_XML_USEBOM_NA • FV_XML_USEBOM_NO • FV_XML_USEBOM_YES When saving as XML, it this is set to FV_XML_USEBOM_YES, FrameMaker writes a byte order mark in the resulting XML.
FP_XmlVersion	StringT	The XML Version that was specified in the XML Declaration when the file was opened. If no XML version was specified, F_ApiGetString() returns an empty string. If this string contains an invalid XML declaration, it will produce a parsing error when this document is saved as XML.
FP_XmlWellFormed	IntT	Indicates whether the source XML qualified as well formed. Can be one of: • FV_XML_WELLFORMED_NA • FV_XML_WELLFORMED_NO • FV_XML_WELLFORMED_YES

Book PDF properties

FO_Book objects have the following properties that provide PDF information

PDF Document Info dictionaries The FP_PDFDocInfo property defines a list of strings to enter in a PDF Document Info dictionary. In PDF, these strings can use Unicode encoding.

The FP_PDFDocInfo property defines a list of strings to enter in a PDF Document Info dictionary. For one dictionary entry, you provide two strings; the first is the entry name, and the second is the entry content. The entry name can be up to 126 bytes; the entry content can be up to 32765 characters.

The entry name is a string of bytes within the ASCII range. For non-printable ASCII, provide Hex codes. To represent a Hex code, precede the code with the "#" character; for example #24. To represent the "#" character, enter #23. Finally, an entry name may not include a byte with a value of zero (#00).

The entry content can include Unicode encoding.

For more information on Acrobat specific functionality see the Acrobat Documentation.

PDF properties The following table lists the PDF properties for books.

Property	Type	Meaning
FP_AcrobatBookmarkDisplayTags	IntT	Retained in Version 6.0 or later for backward compatibility. Use FP_PDFBookmarkDisplay Tags instead.
FP_DocAcrobatColumn ArticleThreads	IntT	Retained in Version 6.0 or later for backward compatibility. Use FP_DocPDFNoArticleThreads instead.
FP_DocAcrobatDefaultsChanged	IntT	Retained in Version 6.0 or later for backward compatibility. Use FP_DocPDFefaults Changed instead.
FP_DocAcrobatElementList	F_Strin gsT	Retained in Version 6.0 or later for backward compatibility. Use FP_DocPDFElementList instead.
FP_DocAcrobatElements	IntT	Retained in Version 6.0 or later for backward compatibility. Use FP_DocPDFElements instead.
FP_DocAcrobatNoArticleThreads	IntT	Retained in Version 6.0 or later for backward compatibility. Use FP_DocPDFNoArticle Threads instead.

Property	Type	Meaning
FP_GenerateAcrobatInfo	IntT	True if Generate Adobe Acrobat Data is on. To generate PDF data, you must set other document print properties as follows: • FP_PrintToFile: True • FP_PrintThumbnails: False • FP_PrintSeps: False • FP_PrintBlankPages: True • FP_PrintLastSheetFirst: False • FP_PrintNumCopies: 1 • FP_PrintOddPages: True • FP_PrintEvenPages: True • FP_PrintScale: 100%
FP_DocPDFColumnArticleThreads	IntT	True if you want separate article threads for each column, False if you want separate article threads for each text frame. Note that FP_DocPDFNoArticleThread must be false.
FP_DocPDFDefaultsChanged	IntT	True if the default heuristics for determining the paragraph level are disabled.
FP_DocPDFElementList	F_StringsT	List of the element tags and context labels to include in bookmarks. FP_DocPDFElementList applies only to structured documents.
FP_DocPDFElements	IntT	True if elements rather than paragraphs are used for bookmarks. FP_DocPDFElements applies only to structured documents.
FP_DocPDFNoArticleThreads	IntT	True if you do not want article threads in the resulting PDF.

Property	Type	Meaning
FP_GenerateAcrobatInfo	IntT	Retained in Version 6.0 or later for backward compatibility. Use FP_GeneratePDFInfo instead.
FP_GeneratePDFInfo	IntT	True if Generate Adobe Acrobat Data is on. To generate PDF data, you must set other document print properties as follows FP_PrintToFile: True FP_PrintThumbnails: False FP_PrintSeps: False FP_PrintBlankPages: True FP_PrintLastSheetFirst: False FP_PrintNumCopies: 1 FP_PrintOddPages: True FP_PrintEvenPages: True FP_PrintScale: 100%
FP_PDFBookmark	BoolT	True if the FrameMaker product will generate bookmarks when saving as PDF.
FP_PDFBookmarksOpenLevel	IntT	The level of bookmarks to have expanded when Acrobat opens the generated PDF document. Can be any integer, or one of the following defined values: • FV_PDFBookmarksOpenDefaultLevel • FV_PDFBookmarksOpenAllLevels • FV_PDFBookmarksOpenNoneLevel If you specify an integer greater than the number of levels in the Bookmarks Settings, FV_PDFBookmarksOpenAllLevels takes effect.
FP_PDFBookmarkDisplayTagsta	IntT	True if Include Paragraph Tags in Bookmark Text is on (the paragraph tag is added before the paragraph text in each bookmark).

Property	Type	Meaning
FP_PDFDestsMarked	IntT	True if the document has paragraphs or elements marked via FP_MarkedForNamed Destination. One of two things must happen in order for this property to be True: The document was created in version 6.0 or later; the document was opened in version 6.0 or later, and the PDF FileSize Optimization client was run over it to mark all paragraphs or elements that are targets of hypertext links. Normally, your client should not set this value.
FP_PDFDistillerAbsent†	IntT	A value of 1 indicates that there is no Acrobat Distiller available.
FP_PDFDocInfo	F_StringsT	A list of strings expressing values to be set in the PDF Document Info dictionary when you save the book as PDF. Each dictionary entry is expressed as a pair of strings; the first string expresses the field name, and the second string expresses the field value.
FP_PDFEndPage	StringT	The last page of the printing page range, in the FrameMaker numbering style
FP_PDFJobOption	StringT	The name of the Distiller Job Options. If the specified name does not exist in the Distiller Job Options list, then the first Distiller Job Option in the list is used.
FP_PDFJobOptionsAbsent†	IntT	A value of 1 indicates that PDF Job Options are not available.
FP_PDFOpenPage	StringT	The PDF page number, in the FrameMaker numbering style, at which Acrobat opens the generated PDF document.
FP_PDFPageHeight	MetricT	Page height for the generated PDF.
FP_PDFPageWidth	MetricT	Page width for the generated PDF.

Property	Type	Meaning
FP_PDFPrintPageRange	IntT	True for generating PDF for the the specified page range; if False, FrameMaker generates PDF for the entire document or book.
FP_PDFSeparateFiles	IntT	True, if a separate PDF file should be generated for each document in a book. This property can be set for single documents, but is ignored in that case.
FP_PDFStartPage	StringT	The first page of the printing page range, in the FrameMaker numbering style.
FP_PDFZoomFactor	MetricT	When FP_PDFZoomType is FV_PDFZoomNone, the zoom percentage of the PDF document (metric 25% to 1600%) If the number is negative or zero, FV_PDFZoomDefault takes effect.
FP_PDFZoomType	IntT	The PDF zoom setting with which Acrobat opens the generated PDF document. Can be one of: • FV_PDFZoomDefault • FV_PDFZoomPage • FV_PDFZoomWidth • FV_PDFZoomHeight • FV_PDFZoomNone If a different value is specified, FV_PDFZoomDefault takes effect.

Property	Type	Meaning
FP_PDFRegistrationMarks	IntT	Registration marks for the generated PDF. May be one of: • FV_PDFRegistrationMarks None • FV_PDFRegistrationMarks Western • FV_PDFRegistrationMarks Tombo
FP_PFDAllNamedDestinations	IntT	True if PDF generated from this book will include Named Destinations for every paragraph and structure element in the book. This results in a larger PDF filesize. If False, the generated PDF will have Named Destinations only for those paragraphs and structure elements that have already been marked with FP_PDFDestinations Marked = True. For documents created in versions earlier than 6.0, True indicates the user has not used the Optimize PDF Size command for the document; normally your client should not change this value to False.

Book print properties

FO_Book objects have the following properties that specify how a book is printed.

Property	Type	Meaning
FP_PrintBlankPages	IntT	True if FP_PageRounding allows empty pages at the end of documents.
FP_PrintCollated	IntT	True if Collate is enabled.
FP_PrintEmulsion	IntT	Direction of print emulsion: FV_EMUL_UP: Emulsion side up FV_EMUL_DOWN: Emulsion side down
FP_PrinterName	StringT	Name of printer. Setting FP_PrinterName on Windows has no effect. When you set FP_PrinterName, you can set the printer to the default printer by specifying NULL.
FP_PrintEvenPages	IntT	True if Print Even-Numbered Pages is enabled.
FP_PrintFileName	StringT	Filename of file to print to. When you set FP_PrintFileName, you can set the filename to the default filename by specifying NULL.
FP_PrintImaging	IntT	Type of print imaging: FV_IMG_POSITIVE FV_IMG_NEGATIVE
FP_PrintLastSheetFirst	IntT	True if Last Sheet First is enabled.
FP_PrintLowRes	IntT	True if Low-Resolution is enabled.
FP_PrintNumCopies	IntT	Number of copies to print.
FP_PrintOddPages	IntT	True if Odd-Numbered Pages is enabled.
FP_PrintPaperHeight	MetricT	Height of paper.
FP_PrintPaperWidth	MetricT	Width of paper.
FP_PrintRegistration Marks	IntT	True if Registration Marks is enabled.
FP_PrintScale	IntT	Scale factor expressed as a percentage of black (metric 0% to 100%)[a]
FP_PrintSeps	IntT	True if Print Separations is enabled.

Property	Type	Meaning
FP_PrintToFile	IntT	True if Print Only to File is enabled.
FP_SkipBlankSeps	IntT	True if Skip Blank Separations is enabled (don't print blank color separations).

a. In the API, most percentages are represented as MetricT fractions. For print scale percentages, the MetricT value 1<<16 or 0x10000 specifies 100%. For more information on MetricT values, see "MetricT values" on page 954.

Book structure properties

The following FO_Book properties apply only to Structured books.

Property	Type	Meaning/possible values
FP_CustomElementList	F_StringsT	List of tags to display when FP_ElementCatalogDisplay is set to FV_ELCAT_CUSTOM.
FP_ElementCatalog†	F_Element CatalogEntriesT	List of elements in Element Catalog.
FP_ElementCatalog Display	IntT	Catalog display options. Show tags for: FV_ELCAT_STRICT: valid children for working start to finish FV_ELCAT_LOOSE: valid children for working in any order FV_ELCAT_CHILDREN: children allowed anywhere in parent FV_ELCAT_ALL: all elements FV_ELCAT_CUSTOM: the list of tags specified by FP_CustomElementList
FP_Element Selection	F_ElementRangeT	The currently selected element range in the book.

Property	Type	Meaning/possible values
FP_FileExtensionOverride	StringT	The file extension to use when saving this document as XML. Typically, this is used to save XHTML with a .htm extension rather than .xml. This setting should be set in the structure application for this document's DOCTYPE, and that setting may be a more reliable source for this value.
FP_FirstFmtChangeListInDoc	F_ObjHandleT	ID of the first format change list in the list of format change lists in the book (FO_FmtChangeList ID).
FP_FirstElementDefInDoc†	F_ObjHandleT	ID of first element definition in book (FO_ElementDef ID).
FP_HighestLevelElement†	F_ObjHandleT	Highest-level element if the book is structured (FO_Element ID).
FP_NewElemAttrDisplay	IntT	Specifies attribute display properties for new elements: FV_ATTR_DISP_NONE: don't display attributes FV_ATTR_DISP_REQSPEC: display required and specified attributes FV_ATTR_DISP_ALL: display all attributes
FP_NewElemAttrEditing	IntT	Specifies when the Edit Attributes dialog box appears for new elements: FV_ATTR_EDIT_NONE FV_ATTR_EDIT_REQUIRED FV_ATTR_EDIT_ALWAYS
FP_SeparateInclusions	IntT	True if inclusions are listed separately in the element catalog.
FP_SgmlApplication	StringT	Retained for backward compatibility. Use FP_StructuredApplication.
FP_UseInitialStructure	IntT	True if Framemaker inserts initial structure for new elements.

Book View Only properties

`FO_Book` objects have the following view-only properties.

Property	Type	Meaning
`FP_BookIsViewOnly`	`IntT`	`True` if the book is view-only.
`FP_ViewOnlyDeadCodes`	`F_UIntsT`	List of F-codes that can't be executed in the book
`FP_ViewOnlyWinBorders`	`IntT`	`True` if the book has normal window borders; `False` if the book's border buttons are suppressed
`FP_ViewOnlyWinPopup`	`IntT`	`True` if the book window pop-up menu is available

FO_BookComponent

An `FO_BookComponent` object represents a book component. `FO_BookComponent` objects have the following properties.

IMPORTANT: *If you change the numbering properties of a* `FO_BookComponent` *object, you must then call* `F_ApiSimpleGenerate()` *or* `F_ApiUpdateBook()` *to update the document associated with the book component.*

Property	Type	Meaning
`FP_BookComponentIsGeneratable`†	`IntT`	`True` if book component is a generated file (`FP_BookComponentType` is not set to `FV_BK_NOT_GENERATABLE`)

The DOCTYPE system identifier for the source XML document.
Books

Property	Type	Meaning
FP_BookComponentType	IntT	Type of book component: FV_BK_INDEX_AUTHOR: index of authors FV_BK_INDEX_FORMATS: index of formats FV_BK_INDEX_MARKER: index of markers FV_BK_INDEX_REFERENCES: index of references FV_BK_INDEX_STAN: standard index FV_BK_INDEX_SUBJECT: subject index FV_BK_LIST_FIGURE: list of figures FV_BK_LIST_FORMATS: list of formats FV_BK_LIST_MARKER: list of markers FV_BK_LIST_MARKER_ALPHA: alphabetical list of markers FV_BK_LIST_PGF: list of paragraphs FV_BK_LIST_PGF_ALPHA: alphabetical list of paragraphs FV_BK_LIST_REFERENCES: list of references FV_BK_LIST_TABLE: list of tables FV_BK_NOT_GENERATABLE: book component is not a generated file FV_BK_TOC: table of contents
FP_BookParent†	F_ObjHandleT	Book that contains the component (FO_Book ID)

Property	Type	Meaning
FP_ChapNumCompute Method	IntT	The component document's chapter numbering type: FV_NUM_CONTINUE: Continue the numbering from the previous chapter. FV_NUM_RESTART: Use the value specified for FP_ChapterNumber. FV_NUM_SAME: Use the same chapter number as for the previous file. FV_NUM_READ_FROM_FILE: Use the numbering properties from the document associated with this book component.
FP_ChapterNumber	IntT	If FP_ChapNumComputeMethod is FV_NUM_RESTART, use this as the chapter number
FP_ChapterNumStyle	IntT	The numbering style; one of: FV_NUMSTYLE_NUMERIC: Arabic. FV_NUMSTYLE_ROMAN_UC: Roman, uppercase. FV_NUMSTYLE_ROMAN_LC: Roman,lowercase. FV_NUMSTYLE_ALPHA_UC: Alphabetic, uppercase. FV_NUMSTYLE_ALPHA_LC: Alphabetic, lowercase. FV_NUMSTYLE_KANJI: Kanji. FV_NUMSTYLE_ZENKAKU: Zenkaku. FV_NUMSTYLE_ZENKAKU_UC: Zenkaku, uppercase. FV_NUMSTYLE_ZENKAKU_LC: Zenkaku, lowercase. FV_NUMSTYLE_KANJI_KAZU: Kazu. FV_NUMSTYLE_DAIJI: Daiji. FV_NUMSTYLE_TEXT: Text.
FP_ChapterNumText	StringT	If FP_ChapNumStyle is FV_NUMSTYLE_TEXT, use this string as the chapter number.

Property	Type	Meaning
FP_ComponentDisplay Text	StringT	Text that displays in the book window when the value of FP_TypeOfDisplayText = FV_BK_TEXT;. To reset FP_ComponentDisplayText so the FrameMaker product automatically updates the text line with normal information, set it to an empty string (""").
FP_ComponentIsSelected	IntT	True if the component is selected in the book window
FP_ExtractTags	F_StringsT	List of paragraph tags or markers type names that are used to set up a generatable file (for example, table of contents, list of figures, standard index or index of authors)
FP_FirstPageNum	IntT	Number for the first page in the component; used when FP_PageNumComputeMethod = FV_NUM_RESTART.
FP_FnFirstNum	StringT	Number for the first footnote in the component; used when FP_FnNumComputeMethod = FV_NUM_RESTART.
FP_FnCustNumString	StringT	Characters for custom document footnote numbers.

Property	Type	Meaning
FP_FnNumStyle	IntT	Footnote numbering style: FV_FN_NUM_NUMERIC: Arabic FV_FN_NUM_ROMAN_UC: Roman uppercase FV_FN_NUM_ROMAN_LC: Roman lowercase FV_FN_NUM_ALPHA_UC: Alphabetic uppercase FV_FN_NUM_ALPHA_LC: Alphabetic lowercase FV_FN_NUM_KANJI: Kanji characters FV_FN_NUM_ZENKAKU: Zenkaku FV_FN_NUM_ZENKAKU_UC: Zenkaku uppercase FV_FN_NUM_ZENKAKU_LC: Zenkaku lowercase FV_FN_NUM_KANJI_KAZU: Kazu FV_FN_NUM_DAIJI: Daiji FV_FN_NUM_CUSTOM: Custom numbering
FP_FnNumCompute Method	IntT	The component document's footnote numbering type: FV_NUM_CONTINUE: Continue the numbering from the previous file. FV_NUM_RESTART: Use the number specified by FP_FnFirstNum. FV_NUM_PER_PAGE: Restart numbering on each page. FV_NUM_READ_FROM_FILE: Use the numbering properties from the document associated with this book component.
FP_GenerateInclude	IntT	True if the document appears in the scroll list of files to be generated by the Generate/Update command for the book

Property	Type	Meaning
FP_ImportFmtInclude	IntT	True if the book component is included in the list of components to be updated with imported formats or element definitions when the user or a client executes Import Formats or Import Element Definitions
FP_InsertLinks	IntT	True if hypertext links are automatically inserted in generated files
FP_Name	StringT	Pathname of document that the component represents
FP_NextComponentInBook	F_ObjHandleT	Next component in the book file (FO_BookComponent ID)
FP_NextSelected ComponentInBook†	F_ObjHandleT	Next selected component in the book window (FO_BookComponent ID)
FP_PageNumbering	IntT	Obsolete for version 6.0 and later; use FP_PageNumComputeMethod: The component document's page numbering type: FV_BK_CONT_PAGE_NUM FV_BK_RESET_PAGE_NUM FV_BK_READ_FROM_FILE
FP_PageNumCompute Method	IntT	The component document's page numbering type: FV_NUM_CONTINUE: Continue the numbering from the previous file. FV_NUM_RESTART: Restart numbering at the value specified by the FP_FirstPageNum property. FV_NUM_READ_FROM_FILE: Use the numbering properties from the document associated with this book component.

Property	Type	Meaning
FP_PageNumStyle	IntT	Page numbering style: FV_PAGE_NUM_NUMERIC: Arabic FV_PAGE_NUM_ROMAN_UC: Roman uppercase FV_PAGE_NUM_ROMAN_LC: Roman lowercase FV_PAGE_NUM_ALPHA_UC: Alphabetic uppercase FV_PAGE_NUM_ALPHA_LC: Alphabetic lowercase FV_PAGE_NUM_KANJI: Kanji characters FV_PAGE_NUM_ZENKAKU: Zenkaku FV_PAGE_NUM_ZENKAKU_UC: Zenkaku uppercase FV_PAGE_NUM_ZENKAKU_LC: Zenkaku lowercase FV_PAGE_NUM_KANJI_KAZU: Kazu FV_PAGE_NUM_DAIJI: Daiji
FP_PagePrefix	StringT	Obsolete for version 6.0 and later; use chapter and volume numbering instead: Page prefix string
FP_PageSide	IntT	Page side to start the component document on: FV_BK_START_FROM_FILE FV_BK_START_NEXT_AVAILABLE FV_BK_START_LEFT FV_BK_START_RIGHT
FP_PageSuffix	StringT	Obsolete for version 6.0 and later; use chapter and volume numbering instead: Page suffix string

Property	Type	Meaning
FP_PgfNumbering	IntT	Obsolete for version 6.0 and later; use FP_PgfNumComputeMethod: The component document's paragraph numbering: FV_BK_CONT_PGF_NUM FV_BK_RESTART_PGF_NUM
FP_PgfNumCompute Method	IntT	The component document's paragraph numbering type: FV_NUM_CONTINUE: Continue the numbering from the previous file. FV_NUM_RESTART: Restart numbering at 1. FV_NUM_READ_FROM_FILE: Use the numbering properties from the document associated with this book component.
FP_PrevComponentInBook	F_ObjHandleT	Previous component in the book file (FO_BookComponent ID)
FP_PrintInclude	IntT	True if the component document is included in list of book files to be printed
FP_TblFnCustNumString	StringT	Characters for custom table footnote numbers.

Property	Type	Meaning
FP_TblFnNumStyle	IntT	Table footnote numbering style: FV_FN_NUM_NUMERIC: Arabic FV_FN_NUM_ROMAN_UC: Roman uppercase FV_FN_NUM_ROMAN_LC: Roman lowercase FV_FN_NUM_ALPHA_UC: Alphabetic uppercase FV_FN_NUM_ALPHA_LC: Alphabetic lowercase FV_FN_NUM_KANJI: Kanji characters FV_FN_NUM_ZENKAKU: Zenkaku FV_FN_NUM_ZENKAKU_UC: Zenkaku uppercase FV_FN_NUM_ZENKAKU_LC: Zenkaku lowercase FV_FN_NUM_KANJI_KAZU: Kazu FV_FN_NUM_DAIJI: Daiji FV_FN_NUM_CUSTOM: Custom numbering
FP_TblFnNumCompute Method	IntT	The component document's table footnote numbering type: FV_NUM_RESTART: Start at 1. FV_NUM_READ_FROM_FILE: Use the numbering properties from the document associated with this book component.
FP_Unique	IntT	UID of the book component

Property	Type	Meaning
FP_VolNumCompute Method	IntT	The component document's volume numbering type: FV_NUM_CONTINUE: Continue the numbering from the previous volume. FV_NUM_RESTART: Use the value specified for FP_VolumeNumber. FV_NUM_SAME: Use the same volume number as for the previous file. FV_NUM_READ_FROM_FILE: Use the numbering properties from the document associated with this book component.
FP_VolumeNumber	IntT	If FP_VolNumComputeMethod is FV_NUM_RESTART, use this as the volume number
FP_VolumeNumStyle	IntT	The numbering style; one of: FV_NUMSTYLE_NUMERIC: Arabic. FV_NUMSTYLE_ROMAN_UC: Roman, uppercase. FV_NUMSTYLE_ROMAN_LC: Roman, lowercase. FV_NUMSTYLE_ALPHA_UC: Alphabetic, uppercase. FV_NUMSTYLE_ALPHA_LC: Alphabetic, lowercase. FV_NUMSTYLE_KANJI: Kanji. FV_NUMSTYLE_ZENKAKU: Zenkaku. FV_NUMSTYLE_ZENKAKU_UC: Zenkaku, uppercase. FV_NUMSTYLE_ZENKAKU_LC: Zenkaku, lowercase. FV_NUMSTYLE_KANJI_KAZU: Kazu. FV_NUMSTYLE_DAIJI: Daiji. FV_NUMSTYLE_TEXT: Text.

Property	Type	Meaning
FP_VolumeNumText	StringT	If FP_VolNumStyle is FV_NUMSTYLE_TEXT, use this string as the chapter number.

Book component structure properties

The following FO_BookComponent properties are used in Structured FrameMaker only.

Property	Type	Meaning/possible values
FP_ComponentElement†	F_ObjHandleT	Component element (FO_Element ID)
FP_ExtractElementTags	F_StringsT	List of element tags that are used to set up a generatable file (for example, table of contents, list of figures, or list of tables)

Character formats

The API uses an `FO_CharFmt` object to represent a character format.

FO_CharFmt

`FO_CharFmt` objects have the following properties.

Property	Type	Meaning
`FP_BkColor`	`F_ObjHandleT`	Text background color
`FP_Capitalization`	`IntT`	The capitalization type: `FV_CAPITAL_CASE_NORM`: normal capitalization (mixed uppercase and lowercase) `FV_CAPITAL_CASE_SMALL`: small caps `FV_CAPITAL_CASE_LOWER`: lowercase letters only `FV_CAPITAL_CASE_UPPER`: uppercase letters only
`FP_ChangeBar`	`IntT`	`True` if Change Bars are on.
`FP_CharTag`	`StringT`	The character format's tag name.
`FP_Color`	`F_ObjHandleT`	Spot color (`FO_Color` ID).
`FP_CombinedFontFamily`	`F_ObjHandleT`	Combined font definition (`FO_CombinedFontDefn`)
`FP_FontAngle`	`IntT`	Font angle (specifies an index into the array of font angles provided by the session property `FP_FontAngleNames`).
`FP_FontEncodingName`†	`StringT`	The font's encoding
`FP_FontFamily`	`IntT`	Font family (specifies an index into the array of font families provided by the session property `FP_FontFamilyNames`).

Property	Type	Meaning
FP_FontPlatformName	StringT	Name that uniquely identifies a font on a specific platform (for more information, see "How the API represents fonts" on page 107 of the *FDK Programmer's Guide*). For combined fonts, this is the Asian font name.
FP_FontPostScript Name	StringT	Name given to a font when it is sent to a PostScript printer (for more information, see "How the API represents fonts" on page 107 of the *FDK Programmer's Guide*). For combined fonts, this is the Asian font name.
FP_UseBkColor	F_ObjHandleT	True: text background color set for this character format (False if AsIS)
FP_WesternFont PlatformName	StringT	Name that uniquely identifies the Roman component of a combined font on a specific platform (for more information, see "How the API represents fonts" on page 107 of the *FDK Programmer's Guide*).
FP_WesternFont PostScriptName	StringT	Name given to the Roman component of a combined font when it is sent to a PostScript printer (for more information, see "How the API represents fonts" on page 107 of the *FDK Programmer's Guide*).
FP_FontSize	MetricT	Font size (2 pt to 400 pt).
FP_FontVariation	IntT	Font variation (specifies an index into the array of font variations provided by the session property FP_FontVariationNames).
FP_FontWeight	IntT	Font weight (specifies an index into the array of font weights provided by the session property FP_FontWeightNames).

Property	Type	Meaning
FP_Language	IntT	Hyphenation and spell-checking language to use: FV_LANG_BRAZILIAN FV_LANG_BRITISH FV_LANG_CANADIAN_FRENCH FV_LANG_CATALAN FV_LANG_DANISH FV_LANG_DUTCH FV_LANG_ENGLISH FV_LANG_FINNISH FV_LANG_FRENCH FV_LANG_GERMAN FV_LANG_ITALIAN FV_LANG_NOLANGUAGE FV_LANG_NORWEGIAN FV_LANG_NYNORSK FV_LANG_PORTUGUESE FV_LANG_SPANISH FV_LANG_SWEDISH FV_LANG_SWISS_GERMAN FV_LANG_JAPANESE FV_LANG_TRADITIONAL_CHINESE FV_LANG_SIMPLIFIED_CHINESE FV_LANG_KOREAN
FP_KernX	MetricT	Horizontal kern value for manual kerning expressed as a percentage of an em (metric –1000% to 1000%).[a] A positive value moves a character right and a negative value moves a character left.
FP_KernY	MetricT	Vertical kern value for manual kerning expressed as a percentage of an em (metric –1000% to 1000%). A positive value moves characters up and a negative value moves characters down.
FP_Name	StringT	The character format's name.
FP_NextCharFmtInDoc†	F_ObjHandleT	Next character format in document (FO_CharFmt ID).

Property	Type	Meaning
FP_Overline	IntT	True if Overline is enabled.
FP_PairKern	IntT	True if Pair Kern is enabled.
FP_Position	IntT	Vertical position of character: FV_POS_NORM: Normal FV_POS_SUB: Subscript FV_POS_SUPER: Superscript
FP_Spread	MetricT	Obsolete property, but still functional. See corresponding "tracking" property below.
FP_Stretch	MetricT	Character stretch (set width) expressed as a percentage of normal stretch for the font (metric –10% to 1000%).
FP_Strikethrough	IntT	True if Strikethrough is enabled.
FP_Tracking	MetricT	Character tracking expressed as a percentage of an em (metric –100% to 1000%).
FP_Underlining	IntT	Underlining type: FV_CB_NO_UNDERLINE FV_CB_SINGLE_UNDERLINE FV_CB_DOUBLE_UNDERLINE FV_CB_NUMERIC_UNDERLINE
FP_UseCapitalization	IntT	True if FP_Capitalization property overrides default; False if As Is setting used.
FP_UseChangeBar	IntT	True if FP_ChangeBar property overrides default; False if As Is setting used.
FP_UseColor	IntT	True if FP_Color property overrides default; False if As Is setting used.
FP_UseFontAngle	IntT	True if FP_FontAngle overrides default; False if As Is setting used.

Property	Type	Meaning
FP_UseFontFamily	IntT	True if FP_FontFamily overrides default; False if As Is setting used.
FP_UseFontSize	IntT	True if FP_FontSize overrides default; False if As Is setting used.
FP_UseFontVariation	IntT	True if FP_FontVariation overrides default; False if As Is setting used.
FP_UseFontWeight	IntT	True if FP_FontWeight overrides default; False if As Is setting used.
FP_UseKernX	IntT	True if FP_KernX overrides default; False if As Is setting used.
FP_UseKernY	IntT	True if FP_KernY overrides default; False if As Is setting used.
FP_UseOverline	IntT	True if FP_Overline property overrides default; False if As Is setting used.
FP_UsePairKern	IntT	True if FP_PairKern property overrides default; False if As Is setting used.
FP_UsePosition	IntT	True if FP_Position overrides default; False if As Is setting used.
FP_UseSpread	IntT	Obsolete property, but still functional. See corresponding "tracking" property below.
FP_UseStretch	IntT	True if FP_Stretch property overrides default, False if As Is setting is used.
FP_UseStrikethrough	IntT	True if FP_Strikethrough property overrides default; False if As Is setting used.

Property	Type	Meaning
FP_UseTracking	IntT	True if FP_Tracking property overrides default; False if As Is setting used.
FP_UseUnderlining	IntT	True if FP_Underlining property overrides default; False if As Is setting used.

a. In the API, most percentages are represented as MetricT fractions. For spread percentages, the MetricT value 1<<16 or 0x10000 specifies 100% or 1. For more information on MetricT values, see "MetricT values" on page 954.

Colors

The API uses an FO_Color object to represent each color in a Frame document. Properties that specify colors specify the IDs of FO_Color objects.

FO_Color

FO_Color objects have the following properties.

Property	Type	Meaning
FP_Black	MetricT	Percentage of black (metric 0% to 100%)[a]
FP_ColorOverprint	IntT	Overprint setting for the color: FV_COLOR_OVERPRINT FV_COLOR_KNOCKOUT
FP_ColorPrintCtl	IntT	Type of color printing used in document: FV_PRINT_SPOT FV_PRINT_PROCESS FV_PRINT_NO
FP_ColorTintPercent	MetricT	The tint percentage (0% to 100%) or FV_COLOR_NOT_TINTED if the color is not a tint. Specifies the percentage of the FP_TintBaseColor to use for tinting.

Property	Type	Meaning
FP_ColorViewCtl	IntT	A 12-bit number for spot color views. The least significant 2 bits are View 1, the next 2 bits are View 2, and so on. For more information, see Figure 3-1 and the text below.
FP_Cyan	MetricT	Percentage of cyan (metric 0% to 100%)
FP_FamilyName	StringT	Color library name. Note that that you must specify the full ink name, including any trademark symbols. For example, use "MUNSELL\xa8 Book of Color" for "MUNSELL® Book of Color."
FP_InkName	StringT	Specifies the name of the color library pigment. Use this instead of FP_Pantone.
FP_Magenta	MetricT	Percentage of magenta (metric 0% to 100%)
FP_Name	StringT	Name of color
FP_NextColorInDoc†	F_ObjHandleT	Next color in document (FO_Color ID)
FP_ReservedColor	IntT	Color names reserved by Frame products: FV_COLOR_NOT_RESERVED FV_COLOR_CYAN FV_COLOR_MAGENTA FV_COLOR_YELLOW FV_COLOR_BLACK FV_COLOR_WHITE FV_COLOR_RED FV_COLOR_GREEN FV_COLOR_BLUE
FP_TintBaseColor	F_ObjHandleT	Color from which the tint is derived (FO_Color ID), or FV_NO_BASE_COLOR if the color is not a tint.
FP_Yellow	MetricT	Percentage of yellow (metric 0% to 100%)

a. In the API, most percentages are represented as MetricT fractions. For color percentages, the MetricT value 1<<16 or 0x10000 specifies 1%. For more information on MetricT values, see "MetricT values" on page 954.

The `FP_ColorViewCtl` property specifies a 12-bit number for color views. The two least significant bits are View 1, the next 2 bits are View 2, and so on, as shown in Figure 3-1.

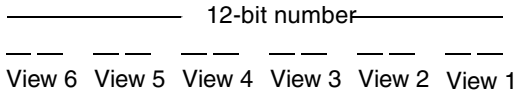


Figure 3-1 *Bit positions representing spot color views*

The values of each 2-bit setting are one of the following:

- `FV_SEP_NORMAL`
- `FV_SEP_NONE`
- `FV_SEP_WHITE`

Columns

The API uses an `FO_SubCol` object to represent each subcolumn in a text frame. `FO_SubCol` objects have the following properties. All of these properties are read-only.

Property	Type	Meaning
`FP_ContentHeight`†	`MetricT`	The distance between the top of the column and the baseline of the last line in the column.
`FP_FirstAFrame`†	`F_ObjHandleT`	First anchored frame in the column (`FO_AFrame` ID).
`FP_FirstCell`†	`F_ObjHandleT`	First table cell in the column (`FO_Cell` ID).
`FP_FirstFn`†	`F_ObjHandleT`	First footnote in the column (`FO_Fn` ID).
`FP_FirstPgf`†	`F_ObjHandleT`	First paragraph in the column (`FO_Pgf` ID).
`FP_FrameParent`†	`F_ObjHandleT`	ID of text frame that contains the column (`FO_TextFrame` ID).
`FP_Height`†	`MetricT`	Column height.
`FP_LastAFrame`†	`F_ObjHandleT`	Last anchored frame in the column (`FO_AFrame` ID).

Property	Type	Meaning
FP_LastCell†	F_ObjHandleT	Last table cell in the column (FO_Cell ID).
FP_LastFn†	F_ObjHandleT	Last footnote in the column (FO_Fn ID).
FP_LastPgf†	F_ObjHandleT	Last paragraph in the column (FO_Pgf ID).
FP_LocX†	MetricT	Offset from left side of the text frame that contains the column.
FP_LocY†	MetricT	Offset from top of text frame that contains the column.
FP_NextSubCol†	F_ObjHandleT	Next column in the text frame that contains the column (FO_SubCol ID).
FP_Overflowed†	IntT	True if the text frame containing the column has Autoconnect turned off and text overflows the column.
FP_ParentTextFrame†	F_ObjHandleT	ID of text frame that contains the column (FO_TextFrame ID).
FP_PrevSubCol†	F_ObjHandleT	Previous column in the text frame that contains the column (FO_SubCol ID).
FP_Unique†	IntT	Text column's UID.
FP_Width†	MetricT	Column width.
Text		To get the text in a column, call F_ApiGetText() with objId set to the column ID. To add text to a column, call F_ApiAddText(). To delete text from a column, call F_ApiDeleteText(). For more information, see "F_ApiAddText()" on page 80, "F_ApiDeleteText()" on page 151, and "F_ApiGetText()" on page 249.

Combined font definitions

The API uses an FO_CombinedFontDefn object to represent each combined font in a document.

`FO_Doc` objects have a property to specify the first combined font in the document's list of combined fonts. To see this property, see "Document object pointer properties" on page 780.

IMPORTANT: *Combined fonts are stored with the document, not with the current session. The session property,* `FP_FontFamilyNames` *returns a list of fonts available for the current session, but it does not include any combined fonts. To get a list of combined font definitions, use* `FP_FirstCombinedFontDefnInDoc` *to get the first combined font definition in the document. From that you can build a list of combined font definitionss using* `FP_NextCombinedFontDefnInDoc`*.*

FO_CombinedFontDefn

`FO_CombinedFontDefn` objects have the following properties.

Property	Type	Meaning
`FP_NextCombinedFontDefnInDoc`†	`IntT`	Next combined font definition instance in the document (`FO_CombinedFontDefn` ID)
`FP_Name`	`StringT`	Name of the combined font.
`FP_BaseFamily`	`IntT`	Asian font family (specifies index into the arrays of font families provided by the session property, `FP_FontFamilyNames`)
`FP_WesternFamily`	`IntT`	Western font family (specifies index into the arrays of font families provided by the session property, `FP_FontFamilyNames`)
`FP_ViewHotspotIndicators`	`BoolT`	Turns on hotspot indicators. Hotspot indicators are small square boxes at the centre of an object to indicate that the object is actually a hotspot.
`FP_WesternSize`	`MetricT`	Scaling factor for Roman text expressed as a percentage of base font size (metric 1% to 1000%)
`FP_WesternShift`	`MetricT`	Baseline offset of Roman text expressed as a percentage of base font size (metric 1% to 1000%)

The DOCTYPE system identifier for the source XML document.

Combined font definitions

Property	Type	Meaning
FP_FontEncodingName†	StringT	Combined font's encoding, based on the FP_BaseFamily
FP_UserString	StringT	

Commands, menus, menu items, and menu item separators

The API uses `FO_Command` objects to represent commands, `FO_Menu` objects to represent menus, and `FO_MenuItemSeparator` objects to represent menu item separators. Menu items are `FO_Command` objects that appear on a menu. Each of these object types has a number of common properties and a number of properties that are specific to it.

Common command, menu, and menu item separator properties

All commands, menus, and menu item separators have the following properties.

Property	Type	Meaning
`FP_Label`	`StringT`	The label the user sees on a menu. The label for menu item separators is read-only; it is always `---`.
`FP_MenuItemIsEnabled`†	`IntT`	`True` if the menu or menu item is enabled or `False` if it is disabled (dimmed).
`FP_Name`†	`StringT`	The command, menu, or menu item separator name.[a]
`FP_NextMenuItemInMenu`	`F_ObjHandleT`	The next menu item, menu, or separator in the menu.
`FP_NextMenuItemInSession`†	`F_ObjHandleT`	The next menu item, menu, or separator in the list of menu items, menus, and separators in the session.
`FP_PrevMenuItemInMenu`	`F_ObjHandleT`	The previous menu item, menu, or separator in the menu.

a. The names for the default, predefined separators are `!Separator`, `!Separator1`, `!Separator2`, `!Separator3`, `!Separator4`, and `!Separator5`.

FO_Command

An `FO_Command` object represents a command or a menu item (a command that appears on a menu). `FO_Command` objects have the following properties.

Property	Type	Meaning
Common command, menu, and menu item separator object properties		See "Common command, menu, and menu item separator properties" on page 760.
`FP_CanHaveCheckMark`	`IntT`	`True` if the menu item can have a check mark. If the menu item is defined by the FrameMaker product, you can get this property, but not set it.
`FP_CheckMarkIsOn`	`IntT`	`True` if the menu item can have a check mark and the check mark is on. If the menu item is defined by the FrameMaker product, you can get this property, but not set it.
`FP_CommandNum`	`IntT`	The integer that you specified for the `cmd` parameter of `F_ApiDefineAndAddCommand()` or `F_ApiDefineCommand()` when you created the command. When the user executes the command, the FrameMaker product passes this integer to your client's `F_ApiCommand()` function. If the menu item is defined by the FrameMaker product, you can get this property, but not set it.
`FP_EnabledWhen`	`IntT`	The context in which the menu item is enabled. For a list of the constants that this field can specify, see the table below. If the menu item is defined by the FrameMaker product, you can get this property, but not set it.
`FP_ExpandOMaticParent`†	`F_ObjHandleT`	If the menu item is an expandomatic menu item, the ID of its virtual parent (`FO_Command` ID).
`FP_Fcode`†	`UIntT`	An f-code that the FrameMaker product executes when the user chooses the menu item or presses the keyboard shortcut.

Property	Type	Meaning
FP_Fcodes†	F_UIntsT	The list of f-codes that the FrameMaker product executes when the user chooses the menu item or presses the keyboard shortcut. Normally, the first f-code in the list is the same as the f-code specified by FP_Fcode.
FP_HasShiftOrUnshift Command	IntT	Specifies whether a command has an accompanying shift command or unshift command: FV_ITEM_HAS_SHIFT_COMMAND FV_ITEM_HAS_UNSHIFT_COMMAND FV_ITEM_HAS_NO_SHIFT_OR_ UNSHIFT_COMMAND
FP_HelpLink	StringT	The hypertext link to call when the user requests context-sensitive help for the command. If you set this property, specify the destination file and an optional page number or linkname. For example, specify foo.doc:lastpage. Do not specify hypertext commands such as gotopage. The FrameMaker product automatically prepends the appropriate hypertext command to the FP_HelpLink string when the user requests context-sensitive help. If the destination file is not in the client directory, the FrameMaker product looks for it in the FrameMaker product help directory. This property is valid only for commands created by clients. It is not valid for commands created directly by FrameMaker products.
FP_KeyboardShortcut Label	StringT	The keyboard shortcut string that appears on the menu. This string need not be one of the actual shortcuts specified by FP_KeyboardShortcuts.

Property	Type	Meaning
FP_KeyboardShortcuts	F_StringsT	The list of keyboard shortcuts the user can press to execute the command. To add a shortcut, append it to the list. The API does not allow you to delete shortcuts from the list.
FP_Labels	F_StringsT	If the command is a menu item, the list of labels it can have in different contexts. If the menu item has only one label in all contexts, FP_Labels specifies only the string for that label. If the menu item has different labels in different contexts, FP_Labels specifies pairs of strings with the following format: *Context*,*Label* where *Label* specifies the menu item label and *Context* specifies the context in which the label appears on the menu. For more information, see "Getting and setting menu item labels" on page 395 of the *FDK Programmer's Guide*.
FP_MenuItemType†	IntT	The type of command or menu item: FV_MENUITEM_FRAME: the command is a menu item defined by the FrameMaker product. FV_MENUITEM_API: the command is a menu item defined by a client. FV_MENUITEM_MACRO: the menu item is not a command; it calls a macro. FV_MENUITEM_EXPANDOMATIC: the menu item is an expandomatic menu item (such as !ShowParagraphTags) defined by the FrameMaker product.
FP_Mode†	IntT	The mode in which keyboard shortcuts are recorded: FV_MODE_ALL FV_MODE_MATH FV_MODE_NONMATH

Property	Type	Meaning
FP_NextCommandIn Session†	F_ObjHandleT	The next command in the list of commands in the session.
FP_ShiftOrUnshift Command	F_ObjHandleT	If FP_HasShiftOrUnshiftCommand is set to FV_ITEM_HAS_SHIFT_COMMAND, the ID of the command to use when the user holds down the Shift key. If FP_HasShiftOrUnshiftCommand is set to FV_ITEM_HAS_UNSHIFT_COMMAND, the ID of the command to use when the user isn't holding down the Shift key.

The following table lists the values FP_EnabledWhen can have and the corresponding contexts in which a menu item is active.

FP_EnabledWhen value	Context in which a menu item is active
FV_ENABLE_ALWAYS_ENABLE	All contexts. This is the default value. If the menu item is disabled, setting FP_EnabledWhen to this value enables it.
FV_ENABLE_ALWAYS_DISABLE	No context. The menu item is disabled. If a menu item is enabled and you set FP_EnabledWhen to this value, it disables and dims the menu item.
FV_ENABLE_BOOK_HAS_ SELECTION	The book contains a selection.
FV_ENABLE_DOC_OR_BOOK_HAS_ SELECTION	A document is in the front, or a book has a selection.
FV_ENABLE_IN_PARA_TEXT	The insertion point or selection is in a paragraph (but not in a math object).
FV_ENABLE_IN_TEXT_LINE	The insertion point or selection is in a graphic text line.
FV_ENABLE_IS_TEXT_SEL	The selection is in a paragraph.
FV_ENABLE_IN_MATH	The insertion point or selection is in a math object.
FV_ENABLE_IN_TEXT	The insertion point or selection is in a graphic text line or a paragraph.

FP_EnabledWhen value	Context in which a menu item is active
FV_ENABLE_OBJ_PROPS	The insertion point is in text, a table, or a math object, or a graphic object is selected.
FV_ENABLE_IN_TABLE	The insertion point or selection is in any part of a table.
FV_ENABLE_IN_TABLE_TITLE	The insertion point or selection is in the table title.
FV_ENABLE_IN_CELL_TEXT	The insertion point or selection is in a table cell.
FV_ENABLE_IS_CELL	A single cell in a table is selected.
FV_ENABLE_IS_CELLS	One or more cells in a table are selected.
FV_ENABLE_IS_TABLE	An entire table is selected.
FV_ENABLE_IS_OBJ	An object is selected.
FV_ENABLE_IS_TEXT_FRAME	A text frame is selected.
FV_ENABLE_IS_OR_IN_FRAME	The selected object is a graphic frame or is in a graphic frame that is not a page frame.
FV_ENABLE_IS_AFRAME	The first selected object is an anchored frame.
FV_ENABLE_IS_GRAPHIC_INSET	The first selected object is a graphic inset.
FV_ENABLE_IS_TEXT_INSET	The first selected object is a text inset.
FV_ENABLE_IN_FLOW	A text frame is selected, or the insertion point or selection is in a paragraph.
FV_ENABLE_COPY	Some text or an object is selected.
FV_ENABLE_COPY_FONT	The insertion point or selection is in the text of a paragraph, a math object, a table, or a text line.
FV_ENABLE_CAN_PASTE	The Clipboard contains an object or text that can be pasted at the insertion point.
FV_ENABLE_IS_VIEW_ONLY	The current document is locked.
FV_ENABLE_NEEDS_DOCP_OR_BOOKP	A document or a book is open.
FV_ENABLE_NEEDS_DOCP_ONLY	A document is open.
FV_ENABLE_NEEDS_BOOKP_ONLY	A book is open.

FO_MenuItemSeparator

An `FO_MenuItemSeparator` object represents a menu item separator. `FO_MenuItemSeparator` objects have only the common properties for commands, menus, and menu item separators.

Property	Type	Meaning
Common command, menu, and menu item separator object properties		See "Common command, menu, and menu item separator properties" on page 760.

FO_Menu

An `FO_Menu` object represents a menu. `FO_Menu` objects have the following properties.

Property	Type	Meaning
Common command, menu, and menu item separator object properties		See "Common command, menu, and menu item separator properties" on page 760.
`FP_FirstMenuItemInMenu`	`F_ObjHandleT`	The first menu item or menu in the menu.
`FP_MenuType`†	`IntT`	Type of menu: `FV_MENU_MENUBAR`: a menu bar defined by the FrameMaker product `FV_MENU_POPUP`: a pop-up menu `FV_MENU_ADHOCRULER`: an ad hoc formatting menu that appears on the ruler `FV_MENU_DEFAULT`: a pull-down or pull-right menu

Conditions

The API uses an `FO_CondFmt` object to represent each conditional text format in a document. It uses FO_AttrCondExpr object to filter output content based on conditional text attributes.

`FO_Doc` objects also have properties that specify how all the condition formats in the document appear. For a list of these properties, see "Document condition properties" on page 797.

FO_AttrCondExpr

An object of type attribute conditional expression. Represents a boolean expression that filters output content. This object has the following properties:

Property	Type	Meaning
`FP_AttrCondExprStr`	`StringT`	Returns or sets the actual Boolean conditional expression
`FP_AttrCondExprIsActive`	`Boolean`	Returns or sets whether the conditional expression is active or not.
`FP_FirstAttrCondExprInDoc`	`StringT`	Returns the handle to the first attribute conditional expression object in doc.
`FP_NextAttrCondExprInDoc`	`StringT`	Returns the handle to the next attribute conditional expression object in document relative to the passed object (of type `FO_AttrCondExpr`)

FO_CondFmt

`FO_CondFmt` objects have the following properties.

Property	Type	Meaning
`FP_BkColor`	`F_ObjHandleT`	Text background color
`FP_CondFmtIsShown`	`IntT`	`True` if the condition is shown. To hide text with a specified condition, set this property and the `FO_Doc` property, `FP_ShowAll`, to `False`.
`FP_Name`	`StringT`	Name of the condition format.

Property	Type	Meaning
FP_NextCondFmtInDoc†	F_ObjHandleT	Next condition format in document (FO_CondFmt ID).
FP_SepOverride	F_ObjHandleT	Color separation format override (FO_Color ID).
FP_StyleOverride	IntT	Style condition indicators for conditional text: FV_CN_CHANGEBAR FV_CN_DOUBLE_UNDERLINE FV_CN_NO_OVERRIDE FV_CN_OVERLINE FV_CN_SINGLE_UNDERLINE FV_CN_STRIKETHROUGH FV_CN_NUMERIC_UNDERLINE FV_CN_NMRIC_AND_CHNGBAR
FP_UseBkColor	F_ObjHandleT	True: text background color set for this character format (False if AsIS)
FP_UseSepOverride	IntT	True if color specified by FP_SepOverride is used instead of default.

Cross-references

The API uses an `FO_XRef` object to represent a cross-reference instance and an `FO_XRefFmt` object to represent a cross-reference format.

FO_XRef

`FO_XRef` objects have the following properties.

Property	Type	Meaning
`FP_Element`†	`F_ObjHandleT`	If the cross-reference is in a Structured document, the ID of the associated element.
`FP_Locked`	`IntT`	`True` if the cross-reference is part of a text inset that retains formatting information from the source document. The cross-reference is not affected by global formatting performed on the document.
`FP_NextXRefInDoc`†	`F_ObjHandleT`	Next cross-reference instance in document (`FO_XRef` ID).
`FP_TextRange`†	`F_TextRangeT`	Text range that the cross-reference instance encompasses.
`FP_Unique`†	`IntT`	The cross-reference's UID.
`FP_XRefFmt`	`F_ObjHandleT`	ID of the cross-reference's format (`FO_XrefFmt` ID).
`FP_XRefFile`	`StringT`	The filename of the file containing the cross-reference source. If the cross-reference source is in the same document as the cross reference, the filename is an empty string ("").
`FP_XrefIs Unresolved`†	`IntT`	True if the FrameMaker product was unable to resolve the cross-reference the last time it updated cross-references. Note that this property is set only when the FrameMaker product updates cross-references. Changes to the document, in and of themselves, do not affect this property.

Property	Type	Meaning
FP_XRefSrcText	StringT	If FP_XRefSrcIsElem is False, the text of the cross-reference source marker; if FP_XRefSrcIsElem is True, a string specifying UID:src_name:text, where UID is the unique ID of the source element, name is the element name, and text is text content of the source element.
FP_XRefSrcIsElem	IntT	True if the cross-reference source is a structural element.

FO_XRefFmt

FO_XRefFmt objects have the following properties.

Property	Type	Meaning
FP_Fmt	StringT	The cross-reference format (a string that specifies text and building blocks)
FP_Name	StringT	The cross-reference format's name
FP_NextXRefFmtInDoc†	F_ObjHandleT	ID of the next cross-reference format (FO_XRefFmt ID)

Dialog boxes

The API uses an FO_DialogResource object to represent a dialog box and the following objects to represent the items in it:

- FO_DlgBox
- FO_DlgButton
- FO_DlgCheckBox
- FO_DlgEditBox
- FO_DlgImage

- FO_DlgLabel
- FO_DlgPopUp
- FO_DlgRadioButton
- FO_DlgScrollBar
- FO_DlgScrollBox
- FO_DlgTriBox

FO_DialogResource

An FO_DialogResource object represents a dialog box. FO_DialogResource objects have the following properties.

Property	Type	Meaning
FP_DockDialog	IntT	Allows setting of the dock location of a modeless dialog. It can have one of the following values: • FV_DIALOG_DOCK_NONE: No docking of dialog; the dialog will be a floating one. • FV_DIALOG_DOCK_LEFT: Dock dialog to the left • FV_DIALOG_DOCK_RIGHT: Dock dialog to the right • FV_DIALOG_DOCK_BOTTOM: Dock dialog to the bottom • FV_DIALOG_DOCK_ALL: Allow docking of dialog to left, right or bottom
FP_DoubleClick†	IntT	True if the dialog box was double-clicked on. The value of this property is valid only when your client gets it in the F_ApiDialogEvent() callback.
FP_Focus	IntT	If set, instead of refocusing on the document after processing a command from the dialog, keeps the focus on the dialog itself.

Property	Type	Meaning
FP_GroupDialog	IntT	FP_GroupDialog property enables the clients to group a modless dialog in a particular group so that when the dialog is launched it opens in the group specified by this property. • FV_DIALOG_GROUP_SPECIAL: Place in special group along with panels like marker, hypertext etc. • FV_DIALOG_GROUP_CATALOGS: Group along with catalogs • FV_DIALOG_GROUP_DESIGNERS: Group along with designers • FV_DIALOG_GROUP_ATTRIBUTES: Group along with panels like attribute-editor • FV_DIALOG_GROUP_PODS: Group with pods • FV_DIALOG_GROUP_PODSRIGHT: Group with pods on right sied e.g FONTPOD • FV_DIALOG_GROUP_EDIT: Place in edit group • FV_DIALOG_GROUP_ALLPANELS: Place with any panel group • FV_DIALOG_GROUP_RMKITS: Place with Rmkits like DITAMap, book etc • FV_DIALOG_GROUP_ALL: Place with any group.

Property	Type	Meaning
FP_HelpLink	StringT	The hypertext link to call when the user requests context-sensitive help for the dialog box. If you set this property, specify the destination file and an optional page number or linkname. For example, specify `foo.doc:lastpage`. Do not specify hypertext commands such as `gotopage`. The FrameMaker product automatically prepends the appropriate hypertext command to the `FP_HelpLink` string when the user requests context-sensitive help. If the destination file is not in the client directory, the FrameMaker product looks for it in the FrameMaker product help directory. If your client has not defined a help link for the dialog box, this string is the default help link to the help main menu. The FrameMaker product includes dialog boxes that are shared by multiple commands. The string for such dialog boxes is prepended with a `Shared db:` statement. .
FP_IsDialogDocked	IntT	Property to determine if the dialog is in docked state or not.
FP_IsDialogVisible	IntT	Property to determine if the dialog is visible or not.
FP_Label	StringT	The dialog box title.
FP_NumItems†	IntT	The number of items in the dialog box.

FP_MaxSize	IntT	Sets maximum size for modeless dialogs (lower word: width, higher word: height). Here is an example of setting maximum size to 230:
		`F_ObjHandleT dlgId = F_ApiOpenResource (FO_DialogResource, (StringT)"dialog");`
		`F_ApiSetInt (FV_SessionId , dlgId, FP_MaxSize, 230);`
		`F_ApiModelessDialog (1, dlgId);`

Property	Type	Meaning

FP_MinSize	IntT	Set minimum size for modeless dialogs (lower word: width, higher word: height). Here is an example of setting minimum size to 230:
		`F_ObjHandleT dlgId = F_ApiOpenResource (FO_DialogResource, (StringT)"dialog");`
		`F_ApiSetInt (FV_SessionId , dlgId, FP_MinSize, 230);`
		`F_ApiModelessDialog (1, dlgId);`

FP_ScreenHeight	IntT	Height of the dialog box window in pixels.
FP_ScreenWidth	IntT	Width of the dialog box window in pixels.
FP_ScreenX	IntT	The offset of the dialog box window in pixels from the left side of the screen (or the left of the FrameMaker product application window on Windows). If you set a value that would result in the dialog box window being off the screen, that value is ignored and the old value is retained.
FP_ScreenY	IntT	The offset of the dialog box window in pixels from the top of the screen (or the top of the FrameMaker product application window on Windows). If you set a value that would result in the dialog box window being off the screen, that value is ignored and the old value is retained.

FO_DlgBox

An `FO_DlgBox` object represents a rectangular box drawn around a set of items in a dialog box. `FO_DlgBox` objects have the following property.

Property	Type	Meaning
FP_Visibility	IntT	`True` if the box is visible

FO_DlgButton

An `FO_DlgButton` object represents a button. `FO_DlgButton` objects have the following properties.

Property	Type	Meaning
`FP_Label`	`StringT`	The label that appears on the button
`FP_Sensitivity`	`IntT`	`True` if the button is enabled; `False` if it is disabled (dimmed)
`FP_State`†	`IntT`	Constant specifying the state of the button: `FV_DlgOptNotActive`: the button was not clicked `FV_DlgOptActive`: the button was clicked
`FP_Visibility`	`IntT`	`True` if the button is visible

FO_DlgCheckBox

An `FO_DlgCheckBox` object represents a checkbox. `FO_DlgCheckBox` objects have the following properties.

Property	Type	Meaning
`FP_Label`	`StringT`	The label that appears next to the checkbox
`FP_Sensitivity`	`IntT`	`True` if the checkbox is enabled; `False` if it is disabled (dimmed)

`FP_State`	`IntT`	Constant specifying the state of the checkbox:
		`FV_DlgOptActive`: the checkbox is `on`
		`FV_DlgOptNotActive`: the checkbox is `off`

`FP_Visibility`	`IntT`	`True` if the checkbox is visible

FO_DlgEditBox

An `FO_DlgEditBox` object represents a text box. `FO_DlgEditBox` objects have the following properties.

Property	Type	Meaning
`FP_Sensitivity`	`IntT`	`True` if the text box is enabled; `False` if it is disabled (dimmed)
`FP_Text`	`StringT`	The text in the text box
`FP_Visibility`	`IntT`	`True` if the text box is visible

FO_DlgImage

An `FO_DlgImage` object represents an image pop-up menu. `FO_DlgImage` objects have the following properties.

Property	Type	Meaning
`FP_Labels`	`F_StringsT`	The list of settings displayed in the image pop-up menu.
`FP_Sensitivity`	`IntT`	`True` if the image pop-up menu is enabled; `False` if it is disabled (dimmed).
`FP_State`	`IntT`	The index (in the list specified by `FP_Labels`) of the chosen setting. If no setting is chosen, it is `-1`.
`FP_Visibility`	`IntT`	`True` if the image pop-up menu is visible.

FO_DlgLabel

An `FO_DlgLabel` object represents a label in a dialog box. `FO_DlgLabel` objects have the following properties.

Property	Type	Meaning
`FP_Label`	`StringT`	The string displayed by the label

Property	Type	Meaning
FP_Sensitivity	IntT	True if the label is enabled; False if it is disabled (dimmed)
FP_Visibility	IntT	True if the label is visible

FO_DlgPopUp

An FO_DlgPopUp object represents a pop-up menu in a dialog box. FO_DlgPopUp objects have the following properties.

Property	Type	Meaning
FP_Labels	F_StringsT	The list of settings displayed in the pop-up menu.
FP_Sensitivity	IntT	True if the pop-up menu is enabled; False if it is disabled (dimmed).
FP_State	IntT	The index (in the list specified by FP_Labels) of the chosen setting. If no setting is chosen, it is -1.
FP_Visibility	IntT	True if the pop-up menu is visible.

FO_DlgRadioButton

An FO_DlgRadioButton object represents a radio button. FO_DlgRadioButton objects have the following properties.

Property	Type	Meaning
FP_Label	StringT	The label that appears next to the radio button.
FP_Sensitivity	IntT	True if the radio button is enabled; False if it is disabled (dimmed).

Property	Type	Meaning
FP_State	IntT	Constant specifying the state of the radio button: FV_DlgOptNotActive: the radio button is not on. FV_DlgOptActive: the radio button is on. Setting FP_State to False has no effect. To turn a radio button off, you must turn on another one of the buttons in the button set.
FP_Visibility	IntT	True if the radio button is visible.

FO_DlgScrollBar

An FO_DlgScrollBar object represents a scroll bar. FO_DlgScrollBar objects have the following properties.

Property	Type	Meaning
FP_IncrVal	IntT	The amount that the scroll bar's thumb box moves when the user clicks on either side of it in scroll bar. For example, if FP_IncrVal is set to 10, the scroll box moves 10 increments to the left when the user clicks to the left of the thumb box.
FP_MaxVal	IntT	The largest value to which the user can drag the scroll bar.
FP_MinVal	IntT	The smallest value to which the user can drag the scroll bar.
FP_Size	MetricT	The scroll bar width if the scroll bar is horizontal or the scroll bar height if the scroll bar is vertical.
FP_Sensitivity	IntT	True if the scroll bar is enabled; False if it is disabled (dimmed).
FP_State	IntT	The value of the scroll bar.
FP_Visibility	IntT	True if the scroll bar is visible.

FO_DlgScrollBox

An `FO_DlgScrollBox` object represents a scroll list. `FO_DlgScrollBox` objects have the following properties.

Property	Type	Meaning
`FP_DoubleClick`	`IntT`	`True` if double-clicking an item has the effect of clicking the default button in the dialog box. The `FP_DoubleClick` property of the `FO_DlgScrollBox` object does not specify whether an item in a scroll list was double-clicked. To determine this, you must get the `FP_DoubleClick` property of the `FO_DialogResource` object in the `F_ApiDialogEvent()` callback. For more information, see "FO_DialogResource" on page 771.
`FP_FirstVis`	`IntT`	The index (in the list specified by `FP_Labels`) of the item that appears at the top of the scroll list.
`FP_Labels`	`F_StringsT`	The list of items in the scroll list.
`FP_NumLines`†	`IntT`	The number of items displayed in the scroll list.
`FP_Sensitivity`	`IntT`	`True` if the scroll list is enabled; `False` if it is disabled (dimmed).
`FP_State`	`IntT`	The index (in the list specified by `FP_Labels`) of the selected item. If no item is selected, it is `-1`.
`FP_Visibility`	`IntT`	`True` if the scroll list is visible.

FO_DlgTriBox

An FO_DlgTriBox object represents a tri-state checkbox (tribox). A tribox is a checkbox that can be on, off, or As Is (grayed or stippled). FO_DlgTriBox objects have the following properties.

Property	Type	Meaning
FP_Label	StringT	The label that appears next to the tribox
FP_Sensitivity	IntT	True if the tribox is enabled; False if it is disabled (dimmed)
FP_State	IntT	Constant specifying the state of the tribox: FV_DlgOptNotActive: the tribox is off FV_DlgOptActive: the tribox is on FV_DlgOptDontCare: the tribox is set to As Is
FP_Visibility	IntT	True if the tribox is visible

Documents

The API uses an FO_Doc object to represent each open document in a FrameMaker product session.

FO_Doc

FO_Doc objects have the following properties.

Document object pointer properties

FO_Doc objects have the following properties that specify the IDs of other objects.

Property	Type	Meaning
FP_BkColor	F_ObjHandleT	Text background color
FP_CurrentPage	F_ObjHandleT	Current page (FO_BodyPage, FO_MasterPage, or FO_RefPage ID)
FP_FirstAttrCondExprInDoc	F_ObjHandleT	First attribute conditional expression object in doc.

Property	Type	Meaning
FP_FirstBodyPageInDoc†	F_ObjHandleT	First body page in the document (FO_BodyPage ID)
FP_FirstCharFmtInDoc†	F_ObjHandleT	First character tag in the list of the document's character tags (FO_CharFmt ID)
FP_FirstColorInDoc†	F_ObjHandleT	First color in the list of document's colors (FO_Color ID)
FP_FirstCombinedFontDefnInDoc†	F_ObjHandleT	First combined font definition in the list of the document's combined font definitions (FO_CombinedFontDefn ID)
FP_FirstCondFmtInDoc†	F_ObjHandleT	First condition tag in the list of the document's condition tags (FO_CondFmt ID)
FP_FirstFlowInDoc†	F_ObjHandleT	First flow in the list of the document's flows (FO_Flow ID)
FP_FirstFnInDoc†	F_ObjHandleT	First footnote in the list of the documents footnotes (FO_Fn ID)
FP_FirstGraphicInDoc†	F_ObjHandleT	First graphic object in the list of the document's graphic objects (FO_*GraphicObject* ID)
FP_FirstMarkerInDoc†	F_ObjHandleT	First marker in the list of the document's markers (FO_Marker ID)
FP_FirstMarkerTypeInDoc†	F_ObjHandleT	First marker type in the list of the document's marker types (FO_MarkerType ID)
FP_FirstMasterPageInDoc†	F_ObjHandleT	First master page in the document (FO_MasterPage ID)
FP_FirstPgfFmtInDoc†	F_ObjHandleT	First paragraph tag in the list of the document's paragraph tags (FO_PgfFmt ID)
FP_FirstPgfInDoc†	F_ObjHandleT	First paragraph (FO_Pgf ID) in the list of the document's paragraphs

Property	Type	Meaning
FP_FirstRefPageInDoc†	F_ObjHandleT	First reference page in the document (FO_RefPage ID)
FP_FirstRubiInDoc†	F_ObjHandleT	First rubi composite in the list of the document's rubi composites (FO_Rubi ID)
FP_FirstRulingFmtInDoc†	F_ObjHandleT	First ruling format in the list of the document's ruling formats (FO_RulingFmt ID)
FP_FirstSelectedTiInDoc†	F_ObjHandleT	First selected text inset in the list of selected text insets in the document (FO_TiApiClient, FO_TiText, FO_TiTextTable, or FO_TiFlow ID)
FP_FirstSelected GraphicInDoc†	F_ObjHandleT	First selected graphic object in the list of selected graphic objects in the document (FO_Graphic ID)
FP_FirstTblFmtInDoc†	F_ObjHandleT	First table format in the list of the document's table formats (FO_TblFmt ID)
FP_FirstTblInDoc†	F_ObjHandleT	First table in the list of the document's tables (FO_Tbl ID)
FP_FirstTiInDoc†	F_ObjHandleT	First text inset in the list of the document's text insets (FO_TiApiClient, FO_TiText, FO_TiTextTable, or FO_TiFlow ID)
FP_FirstVarFmtInDoc†	F_ObjHandleT	First variable format in the list of the document's variable formats (FO_VarFmt ID)
FP_FirstVarInDoc†	F_ObjHandleT	First variable in the list of the document's variables (FO_Var ID)
FP_FirstXRefFmtInDoc†	F_ObjHandleT	First cross-reference format in the list of the document's cross-reference formats (FO_XRefFmt ID)
FP_FirstXRefInDoc†	F_ObjHandleT	First cross-reference in the list of the document's cross-references (FO_XRef ID)

Property	Type	Meaning
FP_HiddenPage†	F_ObjHandleT	Hidden page (FO_HiddenPage ID)
FP_LastBodyPageInDoc†	F_ObjHandleT	Last body page in the document (FO_BodyPage ID)
FP_LastMasterPageInDoc†	F_ObjHandleT	Last master page (FO_MasterPage ID)
FP_LastRefPageInDoc†	F_ObjHandleT	Last reference page in the document (FO_RefPage ID)
FP_LeftMasterPage†	F_ObjHandleT	Left master page (FO_MasterPage ID)
FP_MainFlowInDoc†	F_ObjHandleT	Main flow (FO_Flow ID)
FP_MarkerTypeNames†	F_StringsT	List of markertype names
FP_NextOpenDocIn Session†	F_ObjHandleT	Next open document in the list of open documents in the session (FO_Doc ID)
FP_ReviewerNameList	StringListT	List of all the reviewers who have given review comments (using Track Text Edits) for that document
FP_RightMasterPage†	F_ObjHandleT	Right master page (FO_MasterPage ID)
FP_SelectedTbl†	F_ObjHandleT	If any table cells are selected, the ID of the table containing them (FO_Tbl ID)
FP_ShowElementDescriptive Names	IntT	If true, then show element descriptive names in element catalog, as specified in element definition.
FP_TrackChangesAddedColor	F_ObjHandleT	Text color for the Added text.
FP_TrackChangesDeletedCol or	F_ObjHandleT	Text color for the deleted text.
FP_UseInitialStructureOfA utoInsertedElements	IntT	If true, then auto-insertion rules will be processed recursively. For example, if an element is inserted automatically, and in the element definition auto-insertion rules exist for this element, then those rules will also be processed.

XMP Metadata

FrameMaker supports XMP Metadata, which is a protocol to store information about a file as encoded packets that are available to external applications. This information is similar to the information stored in the PDF Document Info dictionary. However, XMP data can contain fields that have no counterpart in PDF Document Info.

FrameMaker maps the values of string pairs in `FP_PDFDocInfo` to XMP Metadata. If an external application modifies the document's metadata, these mapped fields will be updated in `FP_PDFDocInfo`. Likewise, if you change a field in `FP_PDFDocInfo` via the FDK, then FrameMaker will update the encoded XMP packets to reflect this new information. The following table lists the supported fields. External indicates that the field value can be changed by your client, an external application, or the user interface. Internal indicates a field that FrameMaker maintains—you cannot modify its value.

PDF Field:	XMP Field	Internal/External:
Author	dc:Creator	External
Title	dc:Title	External
Subject	dc:Description	External
Keywords	pdf:Keywords	External
Copyright	dc:Rights	External
Web Statement	xapRights:WebStatement	External
Job Reference	xapBJ:JobRef	External
Marked	xapRights:Marked	External
Creation Date	xap:CreateDate	Internal
ModDate	xap:ModifyDate	Internal
Metadata Date	Metadata Date	Internal
Creator	pdf:CreatorTool	Internal (FrameMaker—not displayed in a dialog box)

You can modify XMP data directly for a document by setting a value to the `FP_FileInfoPacket` document property. The FDK sample clients include a client that reads a text file and sets the file's content to the `FP_FilePacket` property. XMP uses the RDF syntax—see http://www.w3.org/1999/02/22-rdf-syntax-ns#, or print the FP_FilePacket to the console to see the XMP syntax. XMP data can include UNICODE

characters. See “PDF Document Info dictionaries” (below) for information about representing UNICODE in a StringT.

Document PDF properties

`FO_Doc` objects store information to use when you save the document as PDF.

PDF Document Info dictionaries The `FP_PDFDocInfo` property defines a list of strings to enter in a PDF Document Info dictionary. In PDF, these strings can use Unicode encoding.

The `FP_PDFDocInfo` property defines a list of strings to enter in a PDF Document Info dictionary. For one dictionary entry, you provide two strings; the first is the entry name, and the second is the entry content. The entry name can be up to 126 bytes; the entry content can be up to 32765 characters.

The entry name is a string of bytes within the ASCII range. For non-printable ASCII, provide Hex codes. To represent a Hex code, precede the code with the “#” character; for example `#24`. To represent the “#” character, enter `#23`. Finally, an entry name may not include a byte with a value of zero (`#00`).

The entry content can include Unicode encoding.

PDF document properties `FO_Doc` objects have the following properties that provide PDF information:

Property	Type	Meaning
`FP_AcrobatBookmark DisplayTags`	`IntT`	Retained in Version 6.0 or later for backward compatibility. Use `FP_PDFBookmarkDisplayTags` instead.
`FP_DocAcrobatColumn ArticleThreads`	`IntT`	Retained in Version 6.0 or later for backward compatibility. Use `FP_DocPDFColumnArticle Threads` instead.
`FP_DocAcrobatDefaults Changed`	`IntT`	Retained in Version 6.0 or later for backward compatibility. Use `FP_DocPDFDefaultsChanged` instead.
`FP_DocAcrobatElement List`	`F_StringsT`	Retained in Version 6.0 or later for backward compatibility. Use `FP_DocPDFElementList` instead.

Property	Type	Meaning
FP_DocAcrobatElements	IntT	Retained in Version 6.0 or later for backward compatibility. Use FP_DocPDFElements instead.
FP_DocAcrobatNoArticle Threads	IntT	Retained in Version 6.0 or later for backward compatibility. Use FP_DocPDFNoArticleThreads instead.
FP_DocPDFColumn ArticleThreads	IntT	True if you want separate article threads for each column, False if you want separate article threads for each text frame. Note that FP_DocPDFNoArticleThread must be false.
FP_DocPDFDefaults Changed	IntT	True if the default heuristics for determining the paragraph level are disabled.
FP_DocPDFElement List	F_StringsT	List of the element tags and context labels to include in bookmarks. FP_DocPDFElementList applies only to structured documents.
FP_DocPDFElements	IntT	True if elements rather than paragraphs are used for bookmarks. FP_DocPDFElements applies only to structured documents.
FP_DocPDFNoArticle Threads	IntT	True if you do not want article threads in the resulting PDF.

Property	Type	Meaning
FP_GenerateAcrobatInfo	IntT	True if Generate Adobe Acrobat Data is on. To generate PDF data, you must set other document print properties as follows: • FP_PrintToFile: True • FP_PrintThumbnails: False • FP_PrintSeps: False • FP_PrintBlankPages: True • FP_PrintLastSheetFirst: False • FP_PrintNumCopies: 1 • FP_PrintOddPages: True • FP_PrintEvenPages: True FP_PrintScale: 100%
FP_GeneratePDFInfo	IntT	True if Generate Adobe Acrobat Data is on. To generate PDF data, you must set other document print properties as follows FP_PrintToFile: True FP_PrintThumbnails: False FP_PrintSeps: False FP_PrintBlankPages: True FP_PrintLastSheetFirst: False FP_PrintNumCopies: 1 FP_PrintOddPages: True FP_PrintEvenPages: True FP_PrintScale: 100%

Property	Type	Meaning
FP_PDFAllNamed Destinations	IntT	True if PDF generated from this book will include Named Destinations for every paragraph and FrameMaker structure element in the document. This results in a larger PDF filesize. If False, the generated PDF will have Named Destinations only for those paragraphs and structure elements that have already been marked with FP_PDFDestinations Marked = True.
FP_PDFBookmark	BoolT	True if the FrameMaker product will generate bookmarks when saving as PDF.
FP_PDFBookmarksOpenLevel	IntT	The level of bookmarks to have expanded when Acrobat opens the generated PDF document. Can be any integer, or one of the following defined values: • FV_PDFBookmarksOpenDefaultLevel • FV_PDFBookmarksOpenAllLevels • FV_PDFBookmarksOpenNoneLevel If you specify an integer greater than the number of levels in the Bookmarks Settings, FV_PDFBookmarksOpenAllLevels takes effect.
FP_PDFBookmarkDisplay Tags	IntT	True if Include Paragraph Tags in Bookmark Text is on (the paragraph tag is added before the paragraph text in each bookmark).

Property	Type	Meaning
FP_PDFDests Marked	IntT	True if the book has documents with paragraphs or elements marked via FP_MarkedForNamedDestination. One of two things must happen in order for this property to be True: The document was created in version 6.0 or later; the document was opened in version 6.0 or later, and the PDF FileSize Optimization client was run over it to mark all paragraphs or elements that are targets of hypertext links.
FP_PDFDistillerAbsent†	IntT	A value of 1 indicates that there is no Acrobat Distiller available.
FP_PDFDocInfo	F_StringsT	A list of strings expressing values to be set in the PDF Document Info dictionary when you save the document as PDF. Each dictionary entry is expressed as a pair of strings; the first string expresses the entry name, and the second string expresses the entry value.
FP_PDFEndPage	StringT	The last page of the printing page range, in the FrameMaker numbering style
FP_PDFJobOption	StringT	The name of the Distiller Job Options. If the specified name does not exist in the Distiller Job Options list, then the first Distiller Job Option in the list is used.
FP_PDFJobOptionsAbsent†	IntT	A value of 1 indicates that PDF Job Options are not available.
FP_PDFOpenPage	StringT	The PDF page number, in the FrameMaker numbering style, at which Acrobat opens the generated PDF document.
FP_PDFPageHeight	MetricT	Page height for the generated PDF.
FP_PDFPageWidth	MetricT	Page width for the generated PDF.
FP_PDFPrintPageRange	IntT	True for generating PDF for the the specified page range; if False, FrameMaker generates PDF for the entire document or book.

Property	Type	Meaning
FP_PDFRegistrationMarks	IntT	Registration marks for the generated PDF. May be one of: • FV_PDFRegistrationMarksNone • FV_PDFRegistrationMarksWestern • FV_PDFRegistrationMarksTombo
FP_PDFSeparateFiles	IntT	True, if a separate PDF file should be generated for each document in a book. This property can be set for single documents, but is ignored in that case.
FP_PDFStartPage	StringT	The first page of the printing page range, in the FrameMaker numbering style.
FP_PDFStructure	IntT	True if the document will create structured PDF when you save it as PDF. The structure is assigned via the FP_PDFStructureLevel property of FO_PgfFmt objects.
FP_PDFZoomFactor	MetricT	When FP_PDFZoomType is FV_PDFZoomNone, the zoom percentage of the PDF document (metric 25% to 1600%) If the number is negative or zero, FV_PDFZoomDefault takes effect.
FP_PDFZoomType	IntT	The PDF zoom setting with which Acrobat opens the generated PDF document. Can be one of: • FV_PDFZoomDefault • FV_PDFZoomPage • FV_PDFZoomWidth • FV_PDFZoomHeight • FV_PDFZoomNone If a different value is specified, FV_PDFZoomDefault takes effect.

General document properties

FO_Doc objects have the following general properties.

Property	Type	Meaning

FP_BannerTextDisplay	BoolT	Specifies whether banner text should be displayed in a document. Here is an example of usage:
		`F_ApiGetInt(FV_SessionId, F_ObjHandleT docId, FP_BannerTextDisplay);`
		`F_ApiSetInt(FV_SessionId, F_ObjHandleT docId, FP_BannerTextDisplay, True/False);`

FP_ChapNumComputeMethod	IntT	The document's chapter numbering type: FV_NUM_CONTINUE: Continue the numbering from the previous chapter. FV_NUM_RESTART: Use the value specified for FP_ChapterNumber. FV_NUM_SAME: Use the same chapter number as for the previous file.
FP_ChapterNumber	IntT	If FP_ChapNumComputeMethod is FV_NUM_RESTART, use this as the chapter number

Property	Type	Meaning
FP_ChapterNumStyle	IntT	The numbering style; one of: FV_NUMSTYLE_NUMERIC: Arabic. FV_NUMSTYLE_ROMAN_UC: Roman numerals, uppercase. FV_NUMSTYLE_ROMAN_LC: Roman, lowercase. FV_NUMSTYLE_ALPHA_UC: Alphabetic, uppercase. FV_NUMSTYLE_ALPHA_LC: Alphabetic, lowercase. FV_NUMSTYLE_KANJI: Kanji. FV_NUMSTYLE_ZENKAKU: Zenkaku. FV_NUMSTYLE_ZENKAKU_UC: Zenkaku, uppercase. FV_NUMSTYLE_ZENKAKU_LC: Zenkaku, lowercase. FV_NUMSTYLE_KANJI_KAZU: Kazu. FV_NUMSTYLE_DAIJI: Daiji. FV_NUMSTYLE_TEXT: Text.
FP_ChapterNumText	StringT	If FP_ChapNumStyle is FV_NUMSTYLE_TEXT, use this string as the chapter number.
FP_Dictionary	F_StringsT	List of words to accept when spell- checking the document.
FP_DocIsHelp†	Int	True if the document is the FrameMaker product's Help document.
FP_DocIsModified†	IntT	True if document has been modified. While this property is read-only, you can modify a document without setting this property to True by setting FP_Untouchable to True for the document before your client modifies it.
FP_DocIsViewOnly	IntT	True if document is View Only.

Property	Type	Meaning
FP_DocOpenType†	IntT	Type of document the file was opened as: FV_DOC_TYPE_BINARY: Frame binary document FV_DOC_TYPE_TEXT: ASCII text document FV_DOC_TYPE_MIF: MIF document FV_DOC_TYPE_FILTER: a filtered document
FP_DocSaveType†	IntT	Type of document the file is saved as: FV_DOC_TYPE_BINARY: Frame binary document FV_DOC_TYPE_TEXT: ASCII text document FV_DOC_TYPE_MIF: MIF document FV_DOC_TYPE_FILTER: a filtered document
FP_DontUpdateTextInsets	IntT	True if FrameMaker product doesn't automatically update text insets when it opens the document.
FP_DontUpdateXRefs	IntT	True if FrameMaker product doesn't automatically update cross-references when it opens or prints the document.
FP_FormatOverride	IntT	Specfies whether there are format overrides at the current insertion point. If the insertion point is in a text range that has a character format applied to it, FP_FormatOverride is True if (and only if) the text formatting at the insertion point overrides the character format. If the insertion point is in a text range that has does not have a character format applied to it, FP_FormatOverride is True if (and only if) the paragraph containing the insertion point has formatting that overrides the Paragraph Catalog format.
FP_IsOnScreen	IntT	True if document is visible on the screen.
FP_IsDocDitamap†	BoolT	True if the document is a DITA Map.
FP_IsDocDita†	BoolT	True if the document is a DITA document (Map/Topic).

Property	Type	Meaning
FP_LineNumDistance	MetricT	Sets the line number display width, that is, the space in which the line numbers are displayed.
FP_LineNumRestart	IntT	If set, restarts line number display on each page.
FP_LineNumShow	IntT	If set, enables the line number display.
FP_Name†	StringT	Filename of the document.
FP_PageNumCompute Method	IntT	The component document's page numbering type: FV_NUM_CONTINUE: Continue the numbering from the previous file. FV_NUM_RESTART: Restart numbering at the value specified by the FP_FirstPageNum property.
FP_PgfNumCompute Method	IntT	The document's paragraph numbering type: FV_NUM_CONTINUE: Continue the numbering from the previous file. FV_NUM_RESTART: Restart numbering at 1.
FP_PreviewState	Int	Determines the preview state of the document when track changes is enabled. Valid values are: FV_PREVIEW_OFF_TRACK_CHANGE: Turn off preview FV_PREVIEW_ON_ORIGINAL:Turn on preview while showing the original content. Shows the original state of document as if all the tracked changes were rejected FV_PREVIEW_ON_FINAL: Turn on preview while showing the final content. Shows the final state of document as if all the tracked changes were accepted.

Property	Type	Meaning
FP_StatusLine	StringT	String that appears in the document status bar. Note that this property always returns an empty string; it is effectively write-only If you set FP_StatusLine to a string other than an empty string (""), the string will remain in the status bar until you reset it. To reset FP_StatusLine so the FrameMaker product automatically updates the status line with normal status information, set it to an empty string ("").
FP_TextSelection	F_TextRangeT	The currently selected text range or insertion point in the document.
FP_Untouchable	IntT	False by default. Setting this to True allows your client to modify a document without the FrameMaker product setting FP_DocIsModified to True.
FP_ViewOnlyWin Palette	Int	True if document acts like a palette when it is View Only.
FP_VolNumCompute Method	IntT	The document's volume numbering type: FV_NUM_CONTINUE: Continue the numbering from the previous volume. FV_NUM_RESTART: Use the value specified for FP_VolumeNumber. FV_NUM_SAME: Use the same volume number as for the previous file.
FP_VolumeNumber	IntT	If FP_VolNumComputeMethod is FV_NUM_RESTART, use this as the volume number

Property	Type	Meaning
FP_VolumeNumStyle	IntT	The numbering style; one of: FV_NUMSTYLE_NUMERIC: Arabic. FV_NUMSTYLE_ROMAN_UC: Roman numerals, uppercase. FV_NUMSTYLE_ROMAN_LC: Roman numerals,lowercase. FV_NUMSTYLE_ALPHA_UC: Alphabetic, uppercase. FV_NUMSTYLE_ALPHA_LC: Alphabetic, lowercase. FV_NUMSTYLE_KANJI: Kanji. FV_NUMSTYLE_ZENKAKU: Zenkaku. FV_NUMSTYLE_ZENKAKU_UC: Zenkaku, uppercase. FV_NUMSTYLE_ZENKAKU_LC: Zenkaku, lowercase. FV_NUMSTYLE_KANJI_KAZU: Kazu. FV_NUMSTYLE_DAIJI: Daiji. FV_NUMSTYLE_TEXT: Text.
FP_VolumeNumText	StringT	If FP_VolNumStyle is FV_NUMSTYLE_TEXT, use this string as the chapter number.

Document change bar properties

FO_Doc objects have the following properties that specify how change bars appear in a document.

Property	Type	Meaning
FP_AutoChangeBars	IntT	True if Automatic Change Bars is enabled
FP_ChangeBarColor	F_ObjHandleT	The spot color (FO_Color ID)
FP_ChangeBarDistance	MetricT	Distance between change bar and text column

Property	Type	Meaning
FP_ChangeBarPosition	IntT	Position of change bars: FV_CB_COL_LEFT: Left of Column FV_CB_COL_RIGHT: Right of Column FV_CB_COL_NEAREST: Side Nearest to Page Edge FV_CB_COL_FURTHEST: Side Farthest from Page Edge
FP_ChangeBarThickness	MetricT	Width of change bars

Document condition properties

FO_Doc objects have the following properties that specify the conditional text expression and how conditional text appears in a document.

Property	Type	Meaning
FP_BooleanConditionExpression	StringT	Returns or sets the Boolean conditional expression for the document. Different from attribute conditional expression. This expression is created with conditional tags while attribute conditional expression is based on attribute values.
FP_ShowAll	IntT	True if all conditions are displayed
FP_ShowCondIndicators	IntT	True if condition indicators (format overrides) are displayed

Document equation properties

`FO_Doc` objects have the following properties that specify the appearance of all equations in the document.

Property	Type	Meaning
`FP_EqnIntegralSizeLarge`	`MetricT`	Point size of integral symbol in large equations (2 pt to 400 pt)
`FP_EqnIntegralSizeMed`	`MetricT`	Point size of integral symbol in medium equations (2 pt to 400 pt)
`FP_EqnIntegralSizeSmall`	`MetricT`	Point size of integral symbol in small equations (2 pt to 400 pt)
`FP_EqnLevel1SizeLarge`	`MetricT`	Point size of level 1 expression in large equations (2 pt to 400 pt)
`FP_EqnLevel1SizeMed`	`MetricT`	Point size of level 1 expression in medium equations (2 pt to 400 pt)
`FP_EqnLevel1SizeSmall`	`MetricT`	Point size of level 1 expression in small equations (2 pt to 400 pt)
`FP_EqnLevel2SizeLarge`	`MetricT`	Point size of level 2 expression in large equations (2 pt to 400 pt)
`FP_EqnLevel2SizeMed`	`MetricT`	Point size of level 2 expression in medium equations (2 pt to 400 pt)
`FP_EqnLevel2SizeSmall`	`MetricT`	Point size of level 2 expression in small equations (2 pt to 400 pt)
`FP_EqnLevel3SizeLarge`	`MetricT`	Point size of level 3 expression in large equations (2 pt to 400 pt)
`FP_EqnLevel3SizeMed`	`MetricT`	Point size of level 3 expression in medium equations (2 pt to 400 pt)
`FP_EqnLevel3SizeSmall`	`MetricT`	Point size of level 3 expression in small equations (2 pt to 400 pt)
`FP_EqnSigmaSizeLarge`	`MetricT`	Point size of sigma symbol in large equations (2 pt to 400 pt)
`FP_EqnSigmaSizeMed`	`MetricT`	Point size of sigma symbol in medium equations (2 pt to 400 pt)
`FP_EqnSigmaSizeSmall`	`MetricT`	Point size of sigma symbol in small equations (2 pt to 400 pt)

Property	Type	Meaning
FP_Functions	StringT	Character format tag of equation font to apply to Math Functions
FP_Numbers	StringT	Character format tag of equation font to apply to Math Numbers
FP_Strings	StringT	Character format tag of equation font to apply to Math Strings
FP_Symbols	StringT	Character format tag of equation font to apply to Math Symbols
FP_SymbolsList†	F_StringsT	List of math symbol fonts used in Equation Fonts dialog box
FP_Variables	StringT	Character format tag of equation font to apply to Math Variables
FP_VerticalTrackingLarge	MetricT	Vertical tracking in large equations
FP_VerticalTrackingMedium	MetricT	Vertical tracking in medium equations
FP_VerticalTrackingSmall	MetricT	Vertical tracking in small equations

Document hypertext properties

FO_Doc objects have the following properties that specify whether to parse and validate a hypertext command, and indicate the results of the parsing and validation.

To parse a hypertext command, set the value of FP_HypertextCommandText to the command you want to parse. Setting the string executes the parser, and if FP_HypertextDoValidate is true, setting the string executes validation as well.

Property	Type	Meaning
FP_HypertextDoValidate	BoolT	True if the next hypertext string sent to FP_HypertextCommandText will be validated
FP_HypertextCommandText	StringT	The hypertext command to parse. Setting this value executes the parser. If FP_HypertextDoValidate is True, the command will be parsed and validated.

Property	Type	Meaning
FP_HypertextParsedArgs†	StringListT	The value of FP_HypertextCommand, parsed into individual tokens
FP_HypertextParseErr†	IntT	Non-zero if there was a parse error. See "FP_HypertextParseErr" on page 801.
FP_HypertextValidateErr†	IntT	Non-zero if FP_HypertextDoValidate was true and there was a validation error. See "FP_HypertextValidateErr" on page 802.
FP_HypertextParseBadParam†	IntT	If there was a parse error, an index into the FP_HypertextParsedArgs string list.
FP_HypertextParseErrMsg†	StringT	The message FrameMaker generates for a parse error
FP_HypertextParsedCmdCode†	IntT	The FrameMaker hypertext command in FP_HypertextCommandText, as determined by the parser. See "Command codes" on page 803.
FP_HypertextParsedCmdDest†	IntT	For link commands, the destination type in FP_HypertextCommandText, as determined by the parser. See "Link destinations" on page 804.
FP_HypertextParsedCmdDestObjType†	IntT	For links to objects, the type of the object in the target document. See "Link destination object types" on page 805.
FP_HypertextParsedCmdDestObjID†	IntT	For links to objects, the UID of the object in the target document.
FP_HypertextParsedCmdMatrixRows†	IntT	If FP_HypertextParsedCmdCode is FV_CmdMatrix, the number of rows in the matrix.
FP_HypertextParsedCmdMatrixColumns†	IntT	If FP_HypertextParsedCmdCode is FV_CmdMatrix, the number of columns in the matrix.
FP_HypertextParsedLinkName†	StringT	For links to named targets, either the value of a newlink command, or a keyword such as FirstPage or LastPage.

Property	Type	Meaning
FP_HypertextParsedPageName†	StringT	For links to pages, the page number.
FP_HypertextParsedFlowName†	StringT	For popup and matrix commands, the name of the flow (on a reference page) that contains the popup or matrix list of commands.
FP_HypertextParsedClientName†	StringT	For message commands, the name of the API client to receive the message.
FP_HypertextParsedTitle†	StringT	If FP_HypertextParsedCmdCode is FV_CmdAlertTitle, the specified title for the alert box.
FP_HypertextParsedMessage†	StringT	If FP_HypertextParsedCmdCode is FV_CmdAlert, FV_CmdAlerTitle, or FV_CmdMessage, the specified message for the hypertext command.
FP_HypertextParsedDIFileName†	StringT	For links to external files, the absolute path to the target file, expressed in platform independent syntax.

FP_HypertextParseErr The following table shows error codes to which FP_HypertextParseErr can be set:

Value	Meaning
FV_HypertextSyntaxOK	No parse errors
FV_HypertextEmptyCommand	Hypertext string is empty
FV_HypertextUnrecognizedCommand	Cannot map the first keyword to an existing FP_HypertextParsedCmdCode
FV_HypertextMissingArguments	One or more arguments required for the command is missing
FV_HypertextExtraArguments	More than the required number of arguments for the command; extra arguments were ignored
FV_HypertextBadSyntaxPathSpec	File reference expected for this command, but no valid filepath found

Value	Meaning
`FV_HypertextUnanchored PartialPath`	File reference is relative to the current document, but the current document has not been saved; file location could not be calculated
`FV_HypertextHelpDirNot Found`	Default help directory either does not exist (help was not installed) or cannot be found
`FV_HypertextExpectedA NumberParam`	Command expected a number but got text; check `FP_HypertextParseBadParam`

FP_HypertextValidateErr The following table shows error codes to which FP_HypertextValidateErr can be set:

Value	Meaning
`FV_HypertextValid`	No validation errors
`FV_HypertextUses DefaultText`	Default text was found as an argument; are you sure the default text is what you want?
`FV_HypertextFileNot Regular`	The referenced file could not be found, or is not a regular file; for example, it could be a directory name
`FV_HypertextFileNot MakerDoc`	The referenced file is not made by a FrameMaker product
`FV_HypertextCantOpen DestFile`	Can't open the file; perhaps you don't have permission, or the file is locked
`FV_HypertextDestination LinkNotFound`	The referenced file is valid, but can't find the named link within it
`FV_HypertextPageNameNot Found`	The referenced file is valid, but can't find the specified page
`FV_HypertextUnrecognized ObjectType`	The referenced file is valid, but the link is to an object with an unrecognized object type
`FV_HypertextObjectIDNot Found`	A link to an object, but can't find the object
`FV_HypertextBadMatrix Size`	One or both of the matrix dimensions is bad; must be between 1 and 99
`FV_HypertextMatrix CommandInvalid`	One of the cammands in the reference page flow for a matrix command has a parse or validation error
`FV_HypertextFlowMissing Lines`	The reference flow for a matrix or popup command is missing one or more lines

Value	Meaning
FV_HypertextNoNamedFlow	Can't find the named reference flow for a matrix or popup command
FV_HypertextRecursive Flow	The reference flow for a matrix or popup command contains nested popup or matrix commands that name a parent reference flow
FV_HypertextMissingPopup Marker	At least one entry in the popup command's reference flow has no hypertext marker in it
FV_HypertextMissingPopup LabelItem	One entry in the popup command's reference flow has no text in it
FV_HypertextEmptyLineIn MiddleOfPopup	One entry in the popup command's reference flow has no text in it
FV_HypertextCommand IllegalWithinPopup	Invalid command in the popup command's reference flow; for example, matrix or newlink
FV_HypertextFcodeInvalid	Invalid FCode in the hypertext command

Command codes The following table shows the possible values for FP_HypertextParsedCmdCode:

Value	Meaning
FV_CmdError	Parser is in an error state
FV_CmdUnknown	Unknown command
FV_CmdNoop	Command causes no event
FV_CmdAlert	alert command
FV_CmdAlertTitle	alerttitle command
FV_CmdExit	exit commant
FV_CmdGoToLink	gotolink command
FV_CmdGoToLinkFitWin	gotolinkfitwin command
FV_CmdGoToNew	gotonew command
FV_CmdGoToPage	gotopage command
FV_CmdGoToObjectId	gotoObjectId command
FV_CmdGoToObjectIdFitWin	gotoObjectIdfitwin command

Value	Meaning
FV_CmdMatrix	matrix command
FV_CmdMessage	message command
FV_CmdNewLink	newlink command
FV_CmdNextPage	nextpage command
FV_CmdOpenLink	openlink command
FV_CmdOpenLinkFitWin	openlinkfitwin command
FV_CmdOpenNew	opennew command
FV_CmdOpenObjectId	openObjectId command
FV_CmdOpenObjectIdFitWin	openObjectIdfitwin command
FV_CmdOpenPage	openpage command
FV_CmdPopup	popup command
FV_CmdPreviousLink	previouslink command
FV_CmdPreviousLinkFitWin	previouslinkfitwin command
FV_CmdPreviousPage	previouspage command
FV_CmdQuit	quit command
FV_CmdQuitAll	quitall command

Link destinations The following table shows the possible values for FP_HypertextParsedCmdDest:

Value	Meaning
FV_DestNowhere	No destination found
FV_DestMarkerNewLink	Destination is a newlink
FV_DestFirstPage	Destination is the first page of a file
FV_DestLastPage	Destination is the last page of a file
FV_DestPageNum	Destination is a named page (usually a page number)
FV_DestFluidFlow	Destination is to a fluid flow document

Value	Meaning
FV_DestMarker	Destination is a marker
FV_DestObjectId	Destination is an object ID (usually for generated hypertext commands)
FV_DestXRef	Destination is a cross-reference

Link destination object types The following table shows the possible values for FP_HypertextParsedCmdDestObjType:

Value	Meaning
FV_ObjectUnknown	Unknown or invalid object
FV_ObjectMarker	Object is a marker
FV_ObjectPgf	Object is a paragraph
FV_ObjectXref	Object is a cross-reference
FV_ObjectGraphic	Object is a graphic
FV_ObjectElement	Object is an element
FV_ObjectTextInset	Object is a text inset
FV_ObjectDataLink	Object is subscribed data

Document menu properties

FO_Doc objects have the following properties that specify a document's menu bars.

Property	Type	Meaning
FP_MenuBar	F_ObjHandleT	The ID of the document's menu bar (FO_Menu ID)
FP_ViewOnlyMenuBar	F_ObjHandleT	The ID of the document's menu bar when the document is locked (FO_Menu ID)

Document footnote properties

FO_Doc objects have the following properties that specify the appearance of footnotes in a document.

Property	Type	Meaning
FP_FnCustNumString	StringT	Characters for custom document footnote numbers
FP_FnFirstNum	IntT	First document footnote number
FP_FnFmt	StringT	Paragraph tag of footnote
FP_FnHeightPerCol	MetricT	Maximum height allowed for document footnotes (36 pt to 32767 pt)
FP_FnInstancePosition	IntT	Placement of document footnote number in footnote: FV_FN_POS_SUPER: Superscript FV_FN_POS_BASELINE: Baseline FV_FN_POS_SUB: Subscript
FP_FnInstancePrefix	StringT	Prefix to appear before document footnote number in footnote
FP_FnInstanceSuffix	StringT	Suffix to appear after document footnote number in footnote
FP_FnNumCompute Method	IntT	The document's footnote numbering type: FV_NUM_CONTINUE: Continue the numbering from the previous file. FV_NUM_RESTART: Restart numbering at the value specified by the associated FO_Doc object's FP_FnFirstNum property. FV_NUM_PER_PAGE: Restart numbering on each page.
FP_FnNumberingPerPage	IntT	Obsolete for version 6.0 and later; use FP_FnNumComputeMethod instead. Retained for backward compatibility; you do not need to set FAPI_55_BEHAVIOR to use this property. True if document footnote numbering is by page, rather than by flow

Property	Type	Meaning
FP_FnNumStyle	IntT	Document footnote numbering style: FV_FN_NUM_NUMERIC: Arabic FV_FN_NUM_ROMAN_UC: Roman uppercase FV_FN_NUM_ROMAN_LC: Roman lowercase FV_FN_NUM_ALPHA_UC: Alphabetic uppercase FV_FN_NUM_ALPHA_LC: Alphabetic lowercase FV_FN_NUM_KANJI: Kanji characters FV_FN_NUM_ZENKAKU: Zenkaku FV_FN_NUM_ZENKAKU_UC: Zenkaku uppercase FV_FN_NUM_ZENKAKU_LC: Zenkaku lowercase FV_FN_NUM_KANJI_KAZU: Kazu FV_FN_NUM_DAIJI: Daiji FV_FN_NUM_CUSTOM: Custom numbering
FP_FnRefPosition	IntT	Position of footnote reference in document text: FV_FN_POS_SUPER: Superscript FV_FN_POS_BASELINE: Baseline FV_FN_POS_SUB: Subscript
FP_FnRefPrefix	StringT	Prefix to appear before number in document text
FP_FnRefSuffix	StringT	Suffix to appear after number in document text

Document flow properties

The following properties were `FO_Doc` properties in earlier versions of the FDK. In the current version of the FDK, they are `FO_Flow` properties. They are no longer valid `FO_Doc` properties.

Property	Type	Meaning
`FP_MaxInterlinePadding`	`MetricT`	Not a valid `FO_Doc` property in the current version of the FDK. See "FO_Flow" on page 829.
`FP_MaxInterPgfPadding`	`MetricT`	Not a valid `FO_Doc` property in the current version of the FDK. See "FO_Flow" on page 829.

Document page properties

`FO_Doc` objects have the following properties that specify page appearance.

Property	Type	Meaning
`FP_BottomMargin`	`MetricT`	Bottom page margin
`FP_ColGap`	`MetricT`	Size of gap between text columns
`FP_DocIsDoubleSided`	`IntT`	`True` if two-sided page layout
`FP_FirstPageNum`	`IntT`	Page number of first page
`FP_FirstPageVerso`	`IntT`	`False` for right first page; `True` for left first page
`FP_LeftMargin`	`MetricT`	Left page margin
`FP_NumCols`†	`IntT`	Number of columns
`FP_PageHeight`	`MetricT`	Height of the document's pages. Setting this property automatically sets the `FP_PageHeight` property of all of the document's body pages.
`FP_PageNumComputeMethod`	`IntT`	The document's page numbering type: `FV_NUM_CONTINUE`: Continue the numbering from the previous file. `FV_NUM_RESTART`: Restart numbering at the value specified by the associated `FO_Doc` object's `FP_FirstPageNum` property.

Property	Type	Meaning
FP_PageNumStyle	IntT	Page numbering style: FV_PAGE_NUM_NUMERIC: Arabic FV_PAGE_NUM_ROMAN_UC: Roman uppercase FV_PAGE_NUM_ROMAN_LC: Roman lowercase FV_PAGE_NUM_ALPHA_UC: Alphabetic uppercase FV_PAGE_NUM_ALPHA_LC: Alphabetic lowercase FV_PAGE_NUM_KANJI: Kanji characters FV_PAGE_NUM_ZENKAKU: Zenkaku FV_PAGE_NUM_ZENKAKU_UC: Zenkaku uppercase FV_PAGE_NUM_ZENKAKU_LC: Zenkaku lowercase FV_PAGE_NUM_KANJI_KAZU: Kazu FV_PAGE_NUM_DAIJI: Daiji
FP_PageRounding	IntT	How to round pages: FV_PR_DEL_EMPTY: Delete Empty Pages FV_PR_KEEP_NUM_EVEN: Make Page Count Even FV_PR_KEEP_NUM_ODD: Make Page Count Odd FV_PR_DONT_CHANGE: Don't Change Page Count
FP_PageWidth	MetricT	Width of the document's pages. Setting this property automatically sets the FP_PageWidth property of all of the document's body pages.

Property	Type	Meaning
FP_PointPageNumStyle	IntT	Point page numbering style: FV_POINT_PAGE_NUM_NUMERIC: Arabic FV_POINT_PAGE_NUM_ROMAN_UC: Roman uppercase FV_POINT_PAGE_NUM_ROMAN_LC: Roman lowercase FV_POINT_PAGE_NUM_ALPHA_UC: Alphabetic uppercase FV_POINT_PAGE_NUM_ALPHA_LC: Alphabetic lowercase FV_POINT_PAGE_NUM_KANJI: Kanji characters FV_POINT_PAGE_NUM_ZENKAKU: Zenkaku FV_POINT_PAGE_NUM_ZENKAKU_UC: Zenkaku uppercase FV_POINT_PAGE_NUM_ZENKAKU_LC: Zenkaku lowercase FV_POINT_PAGE_NUM_KANJI_KAZU: Kazu FV_POINT_PAGE_NUM_DAIJI: Daiji
FP_RightMargin	MetricT	Right page margin
FP_SmartQuotes	IntT	True if Smart Quotes is enabled
FP_SmartSpaces	IntT	True if Smart Spaces is enabled
FP_TopMargin	MetricT	Top page margin

Document print properties

FO_Doc objects have the following properties that specify how a document is printed.

Property	Type	Meaning
FP_DownloadFonts	IntT	The fonts to download to the printer when printing. The default is FV_PR_DOWNLOAD_NONE. Can be one of: FV_PR_DOWNLOAD_NONE FV_PR_DOWNLOAD_ALL FV_PR_DOWNLOAD_ALL_BUT_STANDARD_13 FV_PR_DOWNLOAD_ALL_BUT_STANDARD_35
FP_PrintBlankPages	IntT	True if FP_PageRounding allows empty page at end of document.
FP_PrintCollated	IntT	True if Collate is enabled.
FP_PrintCols	IntT	If FP_PrintThumbnails is True, the number of columns to print.
FP_PrintEmulsion	IntT	Direction of print emulsion: FV_EMUL_UP: Emulsion side up FV_EMUL_DOWN: Emulsion side down
FP_PrintEndPage	IntT	Number of last page to print. Note that the value of FP_DocFluidFlow must be 0; you can't print a range of pages when a document is in fluid view.
FP_PrintEndPageName	IntT	Page number string for the last page to print; use this when the pages are numbered with a style other than FV_PAGE_NUM_NUMERIC. Note that the value of FP_DocFluidFlow must be 0; you can't print a range of pages when a document is in fluid view.
FP_PrintEndPoint	IntT	Number of last point page to print.
FP_PrinterName	StringT	Name of printer. Setting FP_PrinterName on Windows has no effect. When you set FP_PrinterName, you can set the printer to the default printer by specifying NULL.
FP_PrintEvenPages	IntT	True if Print Even-Numbered Pages is enabled.

Property	Type	Meaning
FP_PrintFileName	StringT	Filename of file to print to. When you set FP_PrintFileName, you can set the filename to the default filename by specifying NULL.
FP_PrintImaging	IntT	Type of print imaging: FV_IMG_POSITIVE FV_IMG_NEGATIVE
FP_PrintLastSheetFirst	IntT	True if Last Sheet First is enabled.
FP_PrintLowRes	IntT	True if Low-Resolution is enabled.
FP_PrintManualFeed	IntT	True if Manual Feed is enabled.
FP_PrintNumCopies	IntT	Number of copies to print.
FP_PrintOddPages	IntT	True if Odd-Numbered Pages is enabled.
FP_PrintPaperHeight	MetricT	Height of paper.
FP_PrintPaperWidth	MetricT	Width of paper.
FP_PrintRegistration Marks	IntT	True if Registration Marks is enabled.
FP_PrintRows	IntT	If FP_PrintThumbnails is True, the number of rows to print.
FP_PrintScale	IntT	Scale factor.
FP_PrintScope	IntT	Pages to print. Note that the value of FP_DocFluidFlow must be 0; you can't print a range of pages when a document is in fluid view. FV_PR_ALL: Print all pages FV_PR_RANGE: Print a range of pages
FP_PrintSeps	IntT	True if Print Separations is enabled.
FP_PrintStartPage	IntT	Number of first page to print. Note that the value of FP_DocFluidFlow must be 0; you can't print a range of pages when a document is in fluid view.

Property	Type	Meaning
FP_PrintStartPage Name	IntT	Page number string for the first page to print; use this when the pages are numbered with a style other than FV_PAGE_NUM_NUMERIC. Note that the value of FP_DocFluidFlow must be 0; you can't print a range of pages when a document is in fluid view.
FP_PrintStartPoint	IntT	Number of first point page to print.
FP_PrintThumbnails	IntT	True if Print Thumbnails is enabled.
FP_PrintToFile	IntT	True if Print Only to File is enabled.
FP_SkipBlankSeps	IntT	True if Skip Blank Separations is enabled (don't print blank color separations).
FP_TomboMarks	BoolT	True if registration marks are enabled, and set to Tombo. When printing Tombo Marks via the API, you must also set FP_PrintRegistrationMarks to True.
FP_Trapwise Compatibility	BoolT	True if Trapwise Compatibility is enabled. Setting this to True automatically sets FP_PrintToFile to True and FP_PrintSep to False.

Document rubi properties

FO_Doc objects have the following properties that specify formatting for rubi composites:

FP_NarrowRubiSpaceForKanji	IntT	Allowable values are: FV_Wide FV_Narrow FV_Proportional
FP_NarrowRubiSpaceForOther	IntT	Allowable values are: FV_Wide FV_Narrow FV_Proportional
FP_RubiAlignAtBoundaries	IntT	TRUE if rubi and oyamoji text should be aligned at line boundaries
FP_RubiOverhang	IntT	TRUE if rubi is allowed to overhang

`FP_RubiSize`	`MetricT`	Scaling factor for rubi text expressed as percentage of the current font size (metric 1% to 1000%). If this property and the `FP_RubiFixedSize` property both have values, the most recently set property value is used.
`FP_RubiFixedSize`	`MetricT`	Fixed size for all rubi text (metric 2pts to 400pts). If this property and the `FP_RubiSize` property both have values, the most recently set property value is used.

`FP_WideRubiSpaceForKanji`	`IntT`	Allowable values are:
		`FV_Wide`
		`FV_Narrow`
		`FV_Proportional`

`FP_WideRubiSpaceForOther`	`IntT`	Allowable values are:
		`FV_Wide`
		`FV_Narrow`
		`FV_Proportional`

Document selection properties

FO_Doc objects have the following properties that specify the selected text or graphics in a document.

Property	Type	Meaning
FP_FirstSelectedGraphicInDoc†	F_ObjHandleT	First selected graphic object in the document's list of selected graphic objects (FO_Graphic ID).
FP_ElementSelection	F_ElementRangeT	The currently selected element range in the document. For information on getting and setting the text selection or insertion point in terms of elements, see "How the API represents the element selection in a structured FrameMaker document" on page 81 of the *FDK Programmer's Guide*.
FP_SelectedTbl†	F_ObjHandleT	If any table cells are selected, the ID of the table containing them (FO_Tbl ID). For information on getting and setting table selections, see "Getting the IDs of selected tables and table rows" on page 285 of the *FDK Programmer's Guide*.
FP_FirstSelectedTiInDoc†	F_ObjHandleT	First selected text inset in the list of selected text insets in the document (FO_TiApiClient, FO_TiText, FO_TiTextTable, or FO_TiFlow ID).
FP_TextSelection	F_TextRangeT	The currently selected text range or insertion point in the document. For information on getting and setting the text selection or insertion point in a document, see "Getting and setting the insertion point or text selection" on page 321 of the *FDK Programmer's Guide*.

For information on how the API represents selected text and graphics, see "How the API represents the selection in a document" on page 80 of the *FDK Programmer's Guide*. For information on getting and setting the structural element selection structured documents, see "Getting and setting the structural element selection" on page 329 of the *FDK Programmer's Guide*.

Document structure properties

FO_Doc objects have the following structure properties, which apply only to structured documents.

Property	Type	Meaning
FP_CustomElementList	F_StringsT	List of tags to display when FP_ElementCatalogDisplay is set to FV_ELCAT_CUSTOM.
FP_DefaultExclusions	F_StringsT	List of exclusions inherited when document is included in a structured book.
FP_DefaultInclusions	F_StringsT	List of inclusions inherited when document is included in a structured book.
FP_ElementBoundary Display	IntT	Element boundary display options: FV_ELEM_DISP_NONE: don't display any element boundaries FV_ELEM_DISP_BRACKETS: display the bracketed boundaries FV_ELEM_DISP_TAGS: display the element tags
FP_ElementCatalog	F_Element CatalogEntriesT	List of elements in Element Catalog.
FP_ElementCatalog Display	IntT	Catalog display options. Show tags for: FV_ELCAT_STRICT: valid children for working start to finish FV_ELCAT_LOOSE: valid children for working in any order FV_ELCAT_CHILDREN: children allowed anywhere in parent FV_ELCAT_ALL: all elements FV_ELCAT_CUSTOM: the list of tags specified by FP_CustomElementList
FP_FirstElementDef InDoc†	F_ObjHandleT	ID of first element definition in the list of element definitions in the document (FO_ElementDef ID).

Property	Type	Meaning
FP_FirstFmtChangeList InDoc	F_ObjHandleT	ID of the first format change list in the list of format change lists in the document (FO_FmtChangeList ID).
FP_MaxBottomMargin	MetricT	Maximum bottom margin allowed in the document.
FP_MaxFirstIndent	MetricT	Maximum first indent allowed in the document.
FP_MaxFontSize	MetricT	Maximum font size allowed in the document.
FP_MaxLeading	MetricT	Maximum leading allowed in the document.
FP_MaxLeftIndent	MetricT	Maximum left indent allowed in the document.
FP_MaxLeftMargin	MetricT	Maximum left margin allowed in the document.
FP_MaxRightIndent	MetricT	Maximum right indent allowed in the document.
FP_MaxRightMargin	MetricT	Maximum right margin allowed in the document.
FP_MaxSpaceAbove	MetricT	Maximum space above paragraph allowed in the document.
FP_MaxSpaceBelow	MetricT	Maximum space below paragraph allowed in the document.
FP_MaxSpread	MetricT	Obsolete property, but still functional. See corresponding "tracking" property below.
FP_MaxStretch	MetricT	Maximum character stretch (set width) expressed as a percentave of normal stretch for the font (metric –10% to 1000%).
FP_MaxTabPosition	MetricT	Maximum tab position allowed in the document.
FP_MaxTracking	MetricT	Maximum character tracking expressed as a percentage of an em.
FP_MaxTopMargin	MetricT	Maximum top margin allowed in the document.

Property	Type	Meaning
FP_MinBottomMargin	MetricT	Minimum bottom margin allowed in the document.
FP_MinFirstIndent	MetricT	Minimum first indent allowed in the document.
FP_MinFontSize	MetricT	Minimum font size allowed in the document.
FP_MinLeading	MetricT	Minimum leading allowed in the document.
FP_MinLeftIndent	MetricT	Minimum left indent allowed in the document.
FP_MinLeftMargin	MetricT	Minimum left margin allowed in the document.
FP_MinRightIndent	MetricT	Minimum right indent allowed in the document.
FP_MinRightMargin	MetricT	Minimum right margin allowed in the document.
FP_MinSpaceAbove	MetricT	Minimum space above paragraph allowed in the document.
FP_MinSpaceBelow	MetricT	Minimum space below paragraph allowed in the document.
FP_MinSpread	MetricT	Obsolete property, but still functional. See corresponding "tracking" property below.
FP_MinStretch	MetricT	Minimum character stretch (set width) expressed as a percentage of normal stretch for the font (metric –10% to 1000%).
FP_MinTabPosition	MetricT	Minimum tab position allowed in the document.
FP_MinTracking	MetricT	Minimum character tracking expressed as a percentage of an em.
FP_MinTopMargin	MetricT	Minimum top margin allowed in the document.

Property	Type	Meaning
FP_NewElemAttrDisplay	IntT	Specifies attribute display properties for new elements: FV_ATTR_DISP_NONE: don't display attributes FV_ATTR_DISP_REQSPEC: display required and specified attributes FV_ATTR_DISP_ALL: display all attributes
FP_NewElemAttrEditing	IntT	Specifies when the Edit Attributes dialog box appears for new elements: FV_ATTR_EDIT_NONE FV_ATTR_EDIT_REQUIRED FV_ATTR_EDIT_ALWAYS
FP_SeparateInclusions	IntT	True if inclusions are listed separately in the Element Catalog.
FP_SgmlApplication	StringT	Retained for backward compatibility. Use FP_StructuredApplication.
FP_UseInitial Structure	IntT	True if FrameMaker inserts initial structure for new elements.

Document table footnote properties

FO_Doc objects have the following properties that specify how table footnotes appear.

Property	Type	Meaning
FP_TblFnCellPosition	IntT	Placement of footnote number in footnote text: FV_FN_POS_SUPER: Superscript FV_FN_POS_BASELINE: Baseline FV_FN_POS_SUB: Subscript
FP_TblFnCellPrefix	StringT	Prefix to appear before table footnote number in table cell
FP_TblFnCellSuffix	StringT	Suffix to appear after table footnote number in table cell

Property	Type	Meaning
FP_TblFnCustNumString	StringT	Characters for custom table footnote numbers
FP_TblFnFmt	StringT	Paragraph tag of table footnote
FP_TblFnNumStyle	IntT	Footnote numbering style for tables in document: FV_FN_NUM_NUMERIC: Arabic FV_FN_NUM_ROMAN_UC: Roman uppercase FV_FN_NUM_ROMAN_LC: Roman lowercase FV_FN_NUM_ALPHA_UC: Alphabetic uppercase FV_FN_NUM_ALPHA_LC: Alphabetic lowercase FV_FN_NUM_KANJI: Kanji characters FV_FN_NUM_ZENKAKU: Zenkaku FV_FN_NUM_ZENKAKU_UC: Zenkaku uppercase FV_FN_NUM_ZENKAKU_LC: Zenkaku lowercase FV_FN_NUM_KANJI_KAZU: Kazu FV_FN_NUM_DAIJI: Daiji FV_FN_NUM_CUSTOM: Custom numbering
FP_TblFnPosition	IntT	Placement of footnote number in text: FV_FN_POS_SUPER: Superscript FV_FN_POS_BASELINE: Baseline FV_FN_POS_SUB: Subscript
FP_TblFnPrefix	StringT	Prefix to appear before number in table footnote
FP_TblFnSuffix	StringT	Suffix to appear after number in table footnote

Document type-in properties

FO_Doc objects have the following type-in properties. These properties specify the text characteristics at the insertion point.

Property	Type	Meaning
FP_Capitalization	IntT	Type of capitalization: FV_CAPITAL_CASE_NORM FV_CAPITAL_CASE_SMALL FV_CAPITAL_CASE_LOWER FV_CAPITAL_CASE_UPPER
FP_ChangeBar	IntT	True if Change Bars are enabled.
FP_CharTag	StringT	Name of character format tag.
FP_Color	F_ObjHandleT	Spot color (FO_Color ID).
FP_CondFmtIsShown	IntT	True if condition is shown.
FP_CombinedFontFamily	F_ObjHandleT	Combined font definition (FO_CombinedFontDefn)
FP_FontEncodingName†	StringT	The font's encoding
FP_FontAngle	IntT	Font angle (specifies an index into the array of font angles provided by the session property, FP_FontAngleNames).
FP_FontFamily	IntT	Font family (specifies an index into the array of font families provided by the session property, FP_FontFamilyNames).
FP_FontPlatformName	StringT	Name that uniquely identifies a font on a specific platform (for more information, see "How the API represents fonts" on page 107 of the *FDK Programmer's Guide*).
FP_FontPostScriptName	StringT	Name given to a font when it is sent to a PostScript printer (for more information, see "How the API represents fonts" on page 107 of the *FDK Programmer's Guide*).
FP_FontSize	MetricT	Font size (2 pt to 400 pt).

Property	Type	Meaning
FP_FontVariation	IntT	Font variation (specifies an index into the array of font variations provided by the session property FP_FontVariationNames).
FP_FontWeight	IntT	Font weight (specifies an index into the array of font weights provided by the session property FP_FontWeightNames).
FP_InCond	F_IntsT	Condition tags that apply to the text (array of FO_CondFmt IDs).
FP_KernX	MetricT	Horizontal kern value for manual kerning expressed as a percentage of an em (metric –100% to 1000%). A positive value moves a character right and a negative value moves a character left.
FP_KernY	MetricT	Vertical kern value for manual kerning expressed as a percentage of an em (metric –100% to 1000%). A positive value moves characters up and a negative value moves characters down.
FP_Overline	IntT	True if Overline style is enabled.
FP_PairKern	IntT	True if Pair Kern is enabled.
FP_Position	IntT	Text position relative to baseline of text: FV_POS_NORM: Normal FV_POS_SUB: Subscript FV_POS_SUPER: Superscript
FP_SepOverride	F_ObjHandleT	Custom color separation override (FO_Color ID).
FP_Spread	MetricT	Obsolete property, but still functional. See corresponding "tracking" property below.
FP_Stretch	MetricT	Character stretch (set width) expressed as a percentage of normal stretch for the font (metric –10% to 1000%).
FP_Strikethrough	IntT	True if Strikethrough style is enabled.

Property	Type	Meaning
FP_Style Overrides†	IntT	Style condition indicators for conditional text: FV_CN_DOUBLE_UNDERLINE FV_CN_NO_OVERRIDE FV_CN_OVERLINE FV_CN_SINGLE_UNDERLINE FV_CN_STRIKETHROUGH
FP_Tracking	MetricT	Character tracking expressed as a percentage of an em (metric –100% to 1000%).
FP_Underlining	IntT	Type of underlining: FV_CB_NO_UNDERLINE FV_CB_SINGLE_UNDERLINE FV_CB_DOUBLE_UNDERLINE FV_CB_NUMERIC_UNDERLINE
FP_UseSepOverride	IntT	True if FP_SepOverride overrides default.

Document typographic properties

FO_Doc objects have the following typographic properties.

Property	Type	Meaning
FP_LineBreakAfter	StringT	Characters at which it is permissible to break lines
FP_SmallCapsSize	MetricT	Scaling factor for small caps expressed as percentage of current font size (metric 1% to 1000%)[a]
FP_SmallCapStretch	MetricT	Character stretch (set width) for small caps expressed as a percentage of normal stretch for the font (metric –10% to 1000%).
FP_SubScriptShift	MetricT	Baseline offset of subscripts expressed as percentage of current font size (metric 1% to 1000%)

Property	Type	Meaning
FP_SubScriptSize	MetricT	Scaling factor for subscripts expressed as percentage of current font size (metric 1% to 1000%)
FP_SubScriptStretch	MetricT	Character stretch (set width) for subscripts expressed as a percentage of normal stretch for the font (metric –10% to 1000%).
FP_SuperScriptShift	MetricT	Baseline offset of superscripts expressed as percentage of current font size (metric 1% to 1000%)
FP_SuperScriptSize	MetricT	Scaling factor for superscripts expressed as percentage of the current font size (metric 1% to 1000%)
FP_SuperScriptStretch	MetricT	Character stretch (set width) for superscripts expressed as a percentage of normal stretch for the font (metric –10% to 1000%).

a. In the API, most percentages are represented as MetricT fractions. For scaling factor and offset percentages, the MetricT value 1<<16 or 0x10000 specifies 100% of the current font size. For more information on MetricT values, see "MetricT values" on page 954.

Document view properties

FO_Doc objects have the following properties that specify how the document appears in the application window. Document must be visible to modify view properties..

Property	Type	Meaning
FP_CurrentPage	F_ObjHandleT	Current page (FO_BodyPage, FO_MasterPage, FO_RefPage ID).
FP_IsIconified	IntT	True if the document window is iconified.
FP_IsInFront	IntT	True if the document window is in front of other windows in the FrameMaker product session.
FP_IsOnScreen	IntT	True if document is visible on the screen.
FP_Label	StringT	The title in the document window title bar.
FP_ScreenHeight	IntT	Height of the document window in pixels.

Property	Type	Meaning
FP_ScreenWidth	IntT	Width of the document window in pixels.
FP_ScreenX	IntT	The offset of the document window in pixels from the left side of the screen (or the left of the FrameMaker product application window on Windows). If you set a value that would result in the document window being off the screen, that value is ignored and the old value is retained.
FP_ScreenY	IntT	The offset of the document window in pixels from the top of the screen (or the top of the FrameMaker product application window on Windows). If you set a value that would result in the document window being off the screen, that value is ignored and the old value is retained.
FP_SnapAngle	MetricT	Angle of rotation for Snap Rotate.
FP_SnapGridUnits	MetricT	Units for Snap Grid Spacing (0 to 32768 pt).
FP_SpotColorView	IntT	Spot color separation view (`0` to `6`). `0` specifies View 1, `1` specifies View 2, and so on.
FP_ViewBorders	IntT	`True` if Borders is enabled.
FP_ViewDisplayUnits	MetricT	The `MetricT` equivalent of one unit in the current Display Units. For example, if Display Units is points, this returns `65536.`
FP_ViewFontSize Units	MetricT	The MetricT equivalent of one unit in the current Font Size Unit. Font size units can be either Points or Q. If Points, this returns `65536.` If Q, this returns `47098`
FP_ViewGrid	IntT	`True` if View Grid is enabled.
FP_ViewGridUnits	MetricT	Units for Grid Lines.
FP_ViewNoGraphics	IntT	`True` if Graphics is not enabled.

Property	Type	Meaning
FP_ViewPage Scrolling	IntT	Page scrolling: FV_SCROLL_VARIABLE FV_SCROLL_HORIZONTAL FV_SCROLL_VERTICAL FV_SCROLL_FACING
FP_ViewRulers	IntT	True if Rulers is enabled.
FP_ViewRulerUnits	MetricT	Units for rulers display.
FP_ViewTextSymbols	IntT	True if Text Symbols is enabled.
FP_Zoom	MetricT	Zoom percentage of document (metric 25% to 1600%).[a]

a. In the API, most percentages are represented as MetricT fractions. For zoom percentages, the MetricT value 1<<16 or 0x10000 specifies 100%. For more information on MetricT values, see "MetricT values" on page 954. For an example of setting the zoom percentage for a document, see "F_ApiSetMetric()" on page 423.

Document View Only properties

FO_Doc objects have the following properties that apply to View Only documents.

Property	Type	Meaning
FP_DocFluidFlow	F_ObjHandleT	The ID of a flow to set to fluid view. To turn this off, set the value to 0.
FP_DocIsViewOnly	IntT	True if the document is a view-only document
FP_ViewOnlyDeadCodes	F_UIntsT	F-codes that can't be executed in the document
FP_ViewOnlyMenuBar	F_ObjHandleT	If the document has a specific menu bar, the ID of the menu bar for the document; otherwise 0

Property	Type	Meaning
FP_ViewOnlySelect	IntT	Specifies whether user can select text or graphics in the document: FV_VOS_USER_ONLY: the user can select text when pressing modifier keys, and link targets (cross-reference sources and newliniks) do not highlight FV_VOS_NONE: the user can't select text, and links targets do not highlight FV_VOS_YES: the user can select text (using modifier keys) and link targets are highlighted
FP_ViewOnlyWinBorders	IntT	True if the document has normal document borders; False if the document scroll bars and border buttons are suppressed
FP_ViewOnlyWinMenubar	IntT	True if the document has a document window menu bar
FP_ViewOnlyWinPalette	IntT	True if the document is a palette
FP_ViewOnlyWinPopup	IntT	True if the document window pop-up menu is available
FP_ViewOnlyXRef	IntT	Specifies the behavior of cross-references in the document: FV_VOX_NOT_ACTIVE: cross-references are not active FV_VOX_GOTO_BEHAVIOR: internal cross-references are active FV_VOX_OPEN_BEHAVIOR: external cross-references are active FV_VOX_ALERT: alert appears when cross-reference is clicked

Document DITA properties

`FO_Doc` objects have the following DITA-related properties.

Property	Type	Meaning
`FP_FirstDITAConrefElementInDoc`	`StringT`	Gets the first Conref element in the document
`FP_DocIsViewOnly`	`StringT`	
`FP_FirstDITALinkElementInDoc`	`StringT`	Gets the first Conref element in the document
`FP_FirstDITATopicrefElementInDoc`	`StringT`	Gets the first Topicref element in the document
`FP_FirstDITATopicsetrefElementInDoc`	`StringT`	Gets the first Topicsetref element in the document
`FP_FirstDITAXrefElementInDoc`	`StringT`	Gets the first Xref element in the document.

Document Key Catalog properties

`FO_Doc` objects have the following Key Catalog properties.

Property	Type	Meaning
`FP_KeyCatalog`	`F_ObjHandleT`	Key catalog being used for the document based on the `KeyCatalogType` setting.

`FP_KeyCatalogType`	`IntT`	Type of key catalog setting for the document. Possible values are:
		`FV_DocKeyCatalogTypeDefault`
		`FV_DocKeyCatalogTypeSpecified`
		`FV_DocKeyCatalogTypeNone`

`FP_SpecifiedKeyCatalog`	`F_ObjHandleT`	Key catalog specified for using for the document.

Elements

See “Structural elements” on page 909.

Flows

The API uses an `FO_Flow` object to represent each flow in the document. For more information on flows see the FDK Programmers Guide.

FO_Flow

`FO_Flow` objects have the following properties.

Property	Type	Meaning
`FP_FlowIsAutoConnect`	`IntT`	`True` if Autoconnect is enabled.
`FP_FlowIsFeathered`	`IntT`	`True` if Feather is enabled.
`FP_FlowIsPostScript`	`IntT`	`True` if flow is PostScript code.
`FP_FlowIsSynchronized`	`IntT`	`True` if Baseline Synchronization is enabled.
`FP_FirstTextFrameIn Flow`†	`F_ObjHandleT`	First text frame in flow (`FO_TextFrame` ID).
`FP_LastTextFrameIn Flow`†	`F_ObjHandleT`	Last text frame in flow (`FO_TextFrame` ID).
`FP_MaxInterlinePadding`	`MetricT`	Maximum interline spacing.
`FP_MaxInterPgfPadding`	`MetricT`	Maximum interparagraph spacing.
`FP_MinHang`	`MetricT`	Maximum character height for synchronization of first line in column. If characters exceed this height, FrameMaker products don’t synchronize the first line.
`FP_Name`	`StringT`	Name of flow tag.
`FP_NextFlowInDoc`†	`F_ObjHandleT`	Next flow in document (`FO_Flow` ID).
`FP_SideHeadRoomInFlow`	`IntT`	`True` if Leave Room for Sideheads in Flow is enabled.

Property	Type	Meaning
FP_Spacing	MetricT	Line spacing for synchronized baselines.
Text		To get the text in a flow, call F_ApiGetText() with objId set to the flow ID. To add text to a flow, call F_ApiAddText(). To delete text from a flow, call F_ApiDeleteText(). For more information, see "F_ApiAddText()" on page 80, "F_ApiDeleteText()" on page 151, and "F_ApiGetText()" on page 249.

Flow structure properties

FO_Flow objects have the following structure properties, which apply only to structured flows in documents.

Property	Type	Meaning
FP_HighestLevelElement†	F_ObjHandleT	Highest-level element in flow (FO_Element ID)

Footnotes

The API uses an FO_Fn object to represent a footnote.

FO_Fn

FO_Fn objects have the following properties.

Property	Type	Meaning
FP_ContentHeight†	MetricT	The distance between the top of the footnote and the baseline of the last line in the footnote
FP_Element†	F_ObjHandleT	If the footnote is in a Structured document, the ID of the element containing the footnote

Property	Type	Meaning
FP_FirstPgf†	F_ObjHandleT	First paragraph in the footnote (FO_Pgf ID)
FP_FnNum†	IntT	Footnote number
FP_InTextFrame†	F_ObjHandleT	Text frame containing the footnote (FO_TextFrame ID)
FP_InTextObj†	F_ObjHandleT	Sub column that contains the footnote (FO_SubCol)
FP_LastPgf†	F_ObjHandleT	Last paragraph in the footnote (FO_Pgf ID)
FP_NextFnInDoc†	F_ObjHandleT	Next footnote (FO_Fn ID) in the document
FP_NextFn†	F_ObjHandleT	Next footnote in the text frame (FO_Fn ID)
FP_Overflowed†	IntT	True if the text in the footnote overflows
FP_PrevFn†	F_ObjHandleT	Previous footnote in the text frame (FO_Fn ID)
FP_TextLoc†	F_TextLocT	Text location of the footnote symbol
FP_Unique†	IntT	Footnote's UID

FO_Doc objects also have properties that specify how all the footnotes in the document appear. For a list of these properties, see "Document footnote properties" on page 806.

Format change lists

The API uses FO_FmtChangeList objects to represent format change lists in a Structured document.

FO_FmtChangeList

Unlike other objects, FO_FmtChangeList objects do not all have the same properties. All FO_FmtChangeList objects have the properties listed in "Format change list general properties," next. However, each FO_FmtChangeList object can have a different combination of the properties listed in the following sections. For more information on change list properties and how to get and set them, see "Manipulating format change list properties" on page 309 of the *FDK Programmer's Guide*.

Format change list general properties

All `FO_FmtChangeList` objects have the following general properties.

Property	Type	Meaning
`FP_BkColor`	`F_ObjHandleT`	Text background color
`FP_FmtChangeList InCatalog`	`IntT`	`True` if the format change list is in the Format Change List Catalog. `False` if it is in an element definition, as part of the text format rules.
`FP_Name`	`StringT`	The name of the format change list if it is in the Format Change List Catalog.
`FP_NextFmtChange ListInDoc`	`F_ObjHandleT`	The ID of the next format change list in the document (`FO_FmtChangeList` ID).
`FP_PgfCatalog Reference`	`StringT`	A paragraph format tag if the format change list specifies one. If this property is set, you can't change any of the other format change list properties, except `FP_Name`.

Format change list advanced properties

`FO_FmtChangeList` objects can have the following advanced properties.

Property	Type	Meaning
`FP_AdjHyphens`	`IntT`	Number of allowable adjacent hyphens
`FP_BottomSeparator`	`StringT`	Name of frame to put below paragraph
`FP_BottomSepAt Indent`	`IntT`	`True` if the position of the frame specified by `FP_BottomSeparator` is at the current left indent
`FP_Hyphenate`	`IntT`	`True` if Automatic Hyphenation is enabled
`FP_HyphMinPrefix`	`IntT`	Minimum number of letters that must precede hyphen
`FP_HyphMinSuffix`	`IntT`	Minimum number of letters that must follow a hyphen
`FP_HyphMinWord`	`IntT`	Minimum length of a hyphenated word
`FP_LetterSpace`	`IntT`	`True` if Word Spacing is enabled

Property	Type	Meaning
FP_MaxSpace	MetricT	Maximum word spacing (percentage of an em space in current font)
FP_MinSpace	MetricT	Minimum word spacing (percentage of an em space in current font)
FP_OptSpace	MetricT	Optimum word spacing
FP_TopSeparator	StringT	Name of frame to put above paragraph
FP_TopSepAtIndent	IntT	True if the position of the frame specified by FP_TopSeparator is at the current left indent

Format change list Asian character spacing properties

FO_FmtChangeList objects have the following Asian character spacing properties.

Property	Type	Meaning
FP_MinJRomSpace	MetricT	Minimum Asian-Roman space
FP_OptJRomSpace	MetricT	Optimum Asian-Roman space
FP_MaxJRomSpace	MetricT	Maximum Asian-Roman space
FP_MinJLetSpace	MetricT	Minimum Asian letter space
FP_OptJLetSpace	MetricT	Optimum Asian letter space
FP_MaxJLetSpace	MetricT	Maximum Asian letter space
FP_YakumonoType	IntT	The Yakumono rules to handle punctuation characters; can be one of FV_FLOATING_YAKUMONO FV_MONOSPACE_YAKUMONO FV_FIXED_YAKUMONO

Format change list autonumber properties

`FO_FmtChangeList` objects can have the following autonumber properties.

Property	Type	Meaning
`FP_AutoNumChar`	`StringT`	Character format for the automatic numbering string specified by `FP_AutoNumString`; `""` if the default character format is used
`FP_AutoNumString`	`StringT`	Autonumber format string (for example, `<n>.<n+>`)
`FP_NumAtEnd`	`IntT`	`True` if numbering position is End of Paragraph; `False` if it is Beginning of Paragraph
`FP_PgfIsAutoNum`	`IntT`	`True` if autonumbering is enabled

Format change list basic properties

`FO_FmtChangeList` objects can have the following basic properties.

Property	Type	Meaning
`FP_FirstIndent`	`MetricT`	The paragraph's first-line left margin, measured from the left side of the current text column (0 cm to 100 cm).
`FP_FirstIndentChange`	`MetricT`	Amount by which to increase or decrease the first-line left margin.
`FP_FirstIndentIs Relative`	`IntT`	`True` if the first indent is relative to the left indent.
`FP_FirstIndentRelPos`	`MetricT`	Position relative to left indent if `FP_FirstIndentIsRelative` is `True`.
`FP_Leading`	`MetricT`	Space below each line in the paragraph.
`FP_LeadingChange`	`MetricT`	Amount by which to increase or decrease the leading.
`FP_LeftIndent`	`MetricT`	The paragraph's left margin, measured from the left side of the current text column (0 cm to 100 cm).

Property	Type	Meaning
FP_LeftIndentChange	MetricT	Amount by which to increase or decrease the left margin.
FP_LineSpacingFixed	IntT	True if the line spacing is fixed.
FP_MoveTabs	MetricT	Amount by which to move all tab positions in the paragraph.
FP_NumTabs	IntT	The number of tabs in the paragraph. To clear all the tabs in the paragraph, set FP_NumTabs to 0.
FP_PgfAlignment	IntT	Horizontal alignment of the paragraph: FV_PGF_LEFT FV_PGF_RIGHT FV_PGF_CENTER FV_PGF_JUSTIFIED
FP_RightIndent	MetricT	The paragraph's right margin, measured from the right side of the current text column.
FP_RightIndentChange	MetricT	Amount by which to increase or decrease the right margin.
FP_SpaceAbove	MetricT	The space above the paragraph.
FP_SpaceAboveChange	MetricT	Amount by which to increase or decrease the space above.
FP_SpaceBelow	MetricT	The space below the paragraph.
FP_SpaceBelowChange	MetricT	Amount by which to increase or decrease the space below.
FP_Tabs	F_TabsT	An array of tab descriptions that specify the positions and types of tab stops in the paragraph. For an example of how to get and set tabs in a paragraph, see "F_ApiSetTabs()" on page 441.

Format change list font properties

`FO_FmtChangeList` objects can have the following font properties.

Property	Type	Meaning

`FP_Capitalization`	`IntT`	Type of capitalization to use:
		`FV_CAPITAL_CASE_NORM`
		`FV_CAPITAL_CASE_SMALL`
		`FV_CAPITAL_CASE_LOWER`
		`FV_CAPITAL_CASE_UPPER`

`FP_ChangeBar`	`IntT`	`True` if Change Bars are on.
`FP_Color`	`F_ObjHandleT`	Spot color (`FO_Color` ID).
`FP_CombinedFontFamily`	`F_ObjHandleT`	Combined font definition (`FO_CombinedFontDefn`)
`FP_FontAngle`	`IntT`	Font angle (specifies an index into the array of font angles provided by the session property `FP_FontAngleNames`).
`FP_FontFamily`	`IntT`	Font family (specifies an index into the array of font families provided by the session property `FP_FontFamilyNames`).

Property	Type	Meaning
FP_Language	IntT	Hyphenation and spell-checking language to use:[a] FV_LANG_BRAZILIAN FV_LANG_BRITISH FV_LANG_CANADIAN_FRENCH FV_LANG_CATALAN FV_LANG_DANISH FV_LANG_DUTCH FV_LANG_ENGLISH FV_LANG_FINNISH FV_LANG_FRENCH FV_LANG_GERMAN FV_LANG_ITALIAN FV_LANG_NOLANGUAGE FV_LANG_NORWEGIAN FV_LANG_NYNORSK FV_LANG_PORTUGUESE FV_LANG_SPANISH FV_LANG_SWEDISH FV_LANG_SWISS_GERMAN FV_LANG_JAPANESE FV_LANG_TRADITIONAL_CHINESE FV_LANG_SIMPLIFIED_CHINESE FV_LANG_KOREAN
FP_FontSize	MetricT	Font size (2 pt to 400 pt).
FP_FontSizeChange	MetricT	Amount by which to increase or decrease the font size.
FP_FontVariation	IntT	Font variation (specifies an index into the array of font variations provided by the session property FP_FontVariationNames).
FP_FontWeight	IntT	Font weight (specifies an index into the array of font weights provided by the session property FP_FontWeightNames).

Property	Type	Meaning
FP_KernX	MetricT	Horizontal kern value for manual kerning expressed as a percentage of an em (metric –100% to 1000%).[b] A positive value moves a character right and a negative value moves a character left.
FP_KernY	MetricT	Vertical kern value for manual kerning expressed as a percentage of an em (metric –100% to 1000%). A positive value moves characters up and a negative value moves characters down.
FP_Overline	IntT	True if Overline is enabled.
FP_PairKern	IntT	True if Pair Kern is enabled.
FP_Position	IntT	Specifies position relative to baseline of text: FV_POS_NORM: Normal FV_POS_SUB: Subscript FV_POS_SUPER: Superscript
FP_Spread	MetricT	Obsolete property, but still functional. See corresponding "tracking" property below.
FP_SpreadChange	MetricT	Obsolete property, but still functional. See corresponding "tracking" property below.
FP_Stretch	MetricT	Character stretch (set width) expressed as a percentage of normal stretch for the font (metric –10% to 1000%).
FP_StretchChange	MetricT	Amount expressed as a percentage (metric –10% to 1000%) by which to increase or decrease the character stretch.
FP_Strikethrough	IntT	True if Strikethrough is enabled.
FP_Tracking	MetricT	Character tracking expressed as a percentage of an em (metric –100% to 1000%).

Property	Type	Meaning
FP_TrackingChange	MetricT	Amount by which to increase or decrease the character tracking.
FP_Underlining	IntT	Type of underlining: FV_CB_NO_UNDERLINE FV_CB_SINGLE_UNDERLINE FV_CB_DOUBLE_UNDERLINE FV_CB_NUMERIC_UNDERLINE

a. The FDE provides a function that returns the language string associated with each of the language constants. For more information, see "F_LanguageString()" on page 576.
b. In the API, most percentages are represented as MetricT fractions. For spread percentages, the MetricT value 1<<16 or 0x10000 specifies 100% or 1. For more information on MetricT values, see "MetricT values" on page 954.

Format change list pagination properties

FO_FmtChangeList objects can have the following pagination properties.

Property	Type	Meaning
FP_BlockLines	IntT	The number of Widow/Orphan lines
FP_KeepWithNext	IntT	True if Keep With Next Paragraph is enabled
FP_KeepWithPrev	IntT	True if Keep With Previous Paragraph is enabled
FP_Placement	IntT	Paragraph placement: FV_PGF_SIDEBODY FV_PGF_SIDEHEAD_TOP FV_PGF_SIDEHEAD_FIRST_BASELINE FV_PGF_SIDEHEAD_LAST_BASELINE FV_PGF_RUN_IN FV_PGF_STRADDLE FV_PGF_STRADDLE_NORMAL_ONLY

Property	Type	Meaning
FP_RunInSeparator	StringT	String for Run-In Head Default Punctuation
FP_Start	IntT	Vertical placement of paragraph: FV_PGF_ANYWHERE FV_PGF_TOP_OF_COL FV_PGF_TOP_OF_PAGE FV_PGF_TOP_OF_LEFT_PAGE FV_PGF_TOP_OF_RIGHT_PAGE

Format change list table cell properties

FO_FmtChangeList objects can have the following table cell properties.

Property	Type	Meaning
FP_CellBottomMargin	MetricT	Amount added to default bottom margin of table cell
FP_CellBottomMargin Change	MetricT	Amount by which to increase or decrease the cell bottom margin
FP_CellBottomMargin Fixed	IntT	True if the cell bottom margin is fixed
FP_CellLeftMargin	MetricT	Amount added to default left margin of table cell
FP_CellLeftMargin Change	MetricT	Amount by which to increase or decrease the cell left margin
FP_CellLeftMargin Fixed	IntT	True if the cell left margin is fixed
FP_CellRightMargin	MetricT	Amount added to default right margin of table cell
FP_CellRightMargin Change	MetricT	Amount by which to increase or decrease the cell right margin
FP_CellRightMargin Fixed	IntT	True if the cell right margin is fixed
FP_CellTopMargin	MetricT	Amount added to default top margin of table cell

Property	Type	Meaning
FP_CellTopMargin Change	MetricT	Amount by which to increase or decrease the cell top margin
FP_CellTopMarginFixed	IntT	True if the cell top margin is fixed
FP_CellVAlignment	IntT	Vertical alignment of a paragraph when it is the first one in a cell: FV_PGF_V_ALIGN_TOP FV_PGF_V_ALIGN_MIDDLE FV_PGF_V_ALIGN_BOTTOM

Format rules

The API uses FO_FmtRule objects to represent format rules in a Structured document. It uses an FO_FmtRuleClause object to represent each rule clause in a format rule.

FO_FmtRule

FO_FmtRule objects have the following properties.

Property	Type	Meaning
FP_CountElements	F_StringsT	If the format rule is a level rule, the list of element tags to count among the element's ancestors; the tags are specified by the *Count ancestors named* element of the format rule.
FP_ElementDef	F_ObjHandleT	If the format rule is not nested, the ID of the element definition that contains it (FO_ElementDef ID).
FP_FmtRuleClause	F_ObjHandleT	If the format rule is nested, the ID of the format rule clause that contains it (FO_FmtRuleClause ID).
FP_FmtRuleClauses†	F_IntsT	IDs of the format rule's format rule clause objects (FO_FmtRuleClause IDs).

Property	Type	Meaning
FP_FmtRuleType	IntT	The format rule's type: FV_CONTEXT_RULE FV_LEVEL_RULE
FP_StopCountingAt	StringT	If the format rule is a level rule, the tag of the element at which to stop counting elements (the tag specified by the *Stop counting at first ancestor named* element.

FO_FmtRuleClause

FO_FmtRuleClause objects have the following properties.

Property	Type	Meaning
FP_ContextLabel	StringT	The context label for generated files. It cannot contain white-space characters or any of these special characters: ( ) & | , * + ? < > % [ ] = ! ; : { } " When a user displays the Set Up dialog box to set up a generated file, the label appears next to elements to which the rule clause applies.
FP_ElemPrefixSuffix	StringT	The text of the prefix or suffix. FP_ElemPrefixSuffix specifies NULL if there is no prefix or suffix.
FP_FmtChangeList†	F_ObjHandleT	If the format rule clause specifies a format change list (FP_RuleClauseType specifies FV_RC_CHANGELIST), the ID of the format change list (FO_FmtChangeList ID). To change the FP_FmtChangeList property, use F_ApiNewFmtRuleObject().

Property	Type	Meaning
FP_FmtChangeListTag	StringT	If the format rule clause specifies a change list (FP_RuleClauseType specifies FV_RC_CHANGELIST_TAG) , the change list's tag.
FP_FmtRule†	F_ObjHandleT	The ID of the format rule containing the format rule clause (FO_FmtRule ID).
FP_FormatTag	StringT	The format tag if the format rule clause specifies one (FP_RuleClauseType specifies FV_RC_TAG). If FP_IsTextRange is True, FP_FormatTag specifies a character format tag; otherwise it specifies a paragraph tag, table tag, marker type, cross-reference format, or equation size.
FP_IsTextRange	IntT	True if the container element is formatted as a text range instead of a paragraph.
FP_RuleClauseType†	IntT	The type of rule clause: FV_RC_TAG FV_RC_CHANGELIST_TAG FV_RC_CHANGELIST FV_RC_SUB_FMTRULE
FP_Specification	StringT	The format clause's context or level specification.
FP_SubFmtRule†	F_ObjHandleT	If the format rule clause contains a nested format rule (FP_RuleClauseType specifies FV_RC_SUB_FMTRULE), the format rule's ID (FO_FmtRule ID). To change the FP_SubFmtRule property, use F_ApiNewFmtRuleObject().

Frames

Frames are a type of graphics. For information on anchored frames, see “FO_AFrame” on page 849. For information on unanchored frames, see “FO_UnanchoredFrame” on page 867.

Graphics

Each type of API graphic object (such as `FO_TextFrame` or `FO_Rectangle`) has a set of properties common to all graphic objects and a set of properties that are specific to it.

Common graphics properties

All API graphic objects, except `FO_AFrame` objects, have the following properties. `FO_AFrame` objects have all the following properties except those marked with a ■ symbol.

Property	Type	Meaning
`FP_Angle`	`MetricT`	Angle of the object's rotation.
`FP_ArrowBaseAngle`	`IntT`	Arrowhead base angle in degrees.
`FP_ArrowLength`	`MetricT`	Arrowhead length (always rounded down to the nearest 1/256 point).
`FP_ArrowScaleFactor`	`MetricT`	Factor by which arrowhead is scaled as line width changes (always rounded down to nearest 1/16 point). It is not used if `FP_ArrowScaleHead` is `False`.
`FP_ArrowScaleHead`	`IntT`	`True` if arrowhead is scaled as the line width changes.
`FP_ArrowTipAngle`	`IntT`	Arrowhead tip angle in degrees.

`FP_ArrowType`	`IntT`	Arrowhead style:
		`FV_ARROW_STICK`
		`FV_ARROW_HOLLOW`
		`FV_ARROW_FILLED`

`FP_BorderWidth`	`MetricT`	Border width (0.015 pt to 360 pt).
`FP_Color`	`F_ObjHandleT`	The spot color (`FO_Color` ID).
`FP_Dash`	`F_MetricsT`	Dash style (see the description below).

`FP_Fill`	`IntT`	The fill pattern (numbers between 0 and 15; see Figure 3-2 on page 848). The FDK provides constants for several fill patterns:
		`FV_FILL_BLACK`
		`FV_FILL_WHITE`
		`FV_FILL_CLEAR`

Property	Type	Meaning
FP_FrameParent	F_ObjHandleT	Frame containing the graphic object (FO_AFrame or FO_UnanchoredFrame ID).
FP_GraphicCantBe Selected	IntT	True if the graphic object can't be selected.
FP_GraphicIs Selected	IntT	True if the graphic object is selected.
■ FP_GroupParent	F_ObjHandleT	Group that the object is in (FO_Group ID). Anchored and unanchored frames do not have this property.
FP_HeadArrow	IntT	Arrowhead at end of line (True if line has arrowhead).
FP_Height	MetricT	Height of object (0.125 pt to 3600 pt).
FP_HotspotCmdStr	StringT	The command string for a hotspot. This must be a valid hypertext command string.
FP_HotspotTitle	StringT	The tooltip text for the hotspot in the outputs that support it (for example HTML). This property is optional.
FP_IsHotspot	BoolT	Whether the object is a hotspot or not. If this property is turned off, the object is no longer a hotspot even if command string is non-empty.
FP_LineCap	IntT	Type of line end: FV_CAP_BUTT FV_CAP_ROUND FV_CAP_SQUARE
FP_LocX	MetricT	Distance from the left side of the parent frame (–216 inches to 216 inches). If the graphic object is an anchored frame, the distance is calculated from the left side of the page frame. You can't set FP_LocX for anchored frames.

Property	Type	Meaning
FP_LocY	MetricT	Distance from the top of the parent frame (–216 inches to 216 inches). If the graphic object is an anchored frame, the distance is calculated from the top of the page frame. You can't set FP_LocY for anchored frames.
FP_NextGraphicInDoc†	F_ObjHandleT	Next graphic object in the document.
■ FP_NextGraphicInFrame	F_ObjHandleT	Next graphic object in the frame.
■ FP_NextGraphicInGroup†	F_ObjHandleT	Next graphic object in the group.
FP_NextSelectedGraphicInDoc†	F_ObjHandleT	Next selected graphic object in document.
FP_Overprint	IntT	Specifies the overprint settings for the object FV_OVERPRINT FV_KNOCKOUT FV_FROMCOLOR
FP_Pen	IntT	The pen pattern (numbers between 0 and 15; see Figure 3-2 on page 848). The FDK provides constants for several pen patterns: FV_FILL_BLACK FV_FILL_WHITE FV_FILL_CLEAR
■ FP_PrevGraphicInFrame	F_ObjHandleT	Previous graphic object in the frame.
■ FP_PrevGraphicInGroup†	F_ObjHandleT	Previous graphic object in the group.
■ FP_Runaround	IntT	Specifies whether text can flow around the object and, if so, whether the text follows the contour of the object or a box shape surrounding the object FV_TR_NONE FV_TR_CONTOUR FV_TR_BBOX

Property	Type	Meaning
■ FP_RunaroundGap	MetricT	If the object is a runaround object, the width of the runaround gap.
FP_TailArrow	IntT	Arrowhead at beginning of line (True if enabled).
FP_TintPercent	MetricT	The tint percentage
FP_Unique†	IntT	The graphic object's UID.
FP_Width	MetricT	Width of object (0.125 pt to 3600 pt).

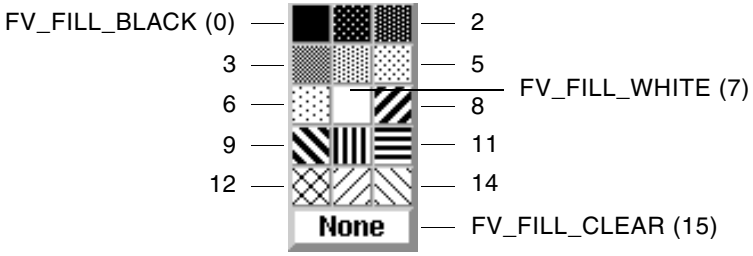


Figure 3-2 *Pen and fill patterns*

The FP_Dash property specifies a dash pattern that is repeated for the length of an object's border. The pattern is stored in an F_MetricsT structure. The 0th element of the F_MetricsT.F_MetricsT_val array stores the length of the first dash; the 1st element stores the following space; the 2nd element stores the next dash; and so on for an even number of elements. For an example of how to set a dashed line for an object, see "F_ApiSetMetrics()" on page 426.

FO_AFrame

An `FO_AFrame` object represents an anchored frame. `FO_AFrame` objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845
`FP_AFrameIsCropped`	`IntT`	`True` if Cropped is enabled
`FP_AFrameIsFloating`	`IntT`	`True` if Floating is enabled

`FP_Alignment`	`IntT`	Type of alignment:
		`FV_ALIGN_CENTER`
		`FV_ALIGN_INSIDE`
		`FV_ALIGN_OUTSIDE`
		`FV_ALIGN_LEFT`
		`FV_ALIGN_RIGHT`

`FP_AnchorType`	`IntT`	Where frame is anchored:
		`FV_ANCHOR_BELOW`
		`FV_ANCHOR_BOTTOM`
		`FV_ANCHOR_INLINE`
		`FV_ANCHOR_RUN_INTO_PARAGRAPH`
		`FV_ANCHOR_SUBCOL_FARTHEST`
		`FV_ANCHOR_SUBCOL_INSIDE`
		`FV_ANCHOR_SUBCOL_LEFT`
		`FV_ANCHOR_SUBCOL_NEAREST`
		`FV_ANCHOR_SUBCOL_OUTSIDE`
		`FV_ANCHOR_SUBCOL_RIGHT`
		`FV_ANCHOR_TEXTFRAME_FARTHEST`
		`FV_ANCHOR_TEXTFRAME_INSIDE`
		`FV_ANCHOR_TEXTFRAME_LEFT`
		`FV_ANCHOR_TEXTFRAME_NEAREST`
		`FV_ANCHOR_TEXTFRAME_OUTSIDE`
		`FV_ANCHOR_TEXTFRAME_RIGHT`
		`FV_ANCHOR_TOP`

`FP_BaselineOffset`	`MetricT`	Baseline offset

Property	Type	Meaning
FP_Element†	F_ObjHandleT	If the anchored frame is in a structured flow in a document, the ID of the element containing the anchored frame
FP_FirstGraphicInFrame†	F_ObjHandleT	First object in frame
FP_InTextFrame†	F_ObjHandleT	Text frame in which anchored frame appears (FO_TextFrame ID)
FP_InTextObj†	F_ObjHandleT	Column or text frame in which anchored frame appears (FO_SubCol or FO_TextFrame ID)
FP_LastGraphicInFrame†	F_ObjHandleT	Last object in frame
FP_NextAFrame†	F_ObjHandleT	Next anchored frame in text frame (FO_AFrame ID)
FP_PrevAFrame†	F_ObjHandleT	Previous anchored frame in text frame (FO_AFrame ID)
FP_SideOffset	MetricT	Near side offset
FP_TextLoc†	F_TextLocT	Text location of the anchor symbol

For different anchored frame types (FP_AnchorType), certain properties of the FO_AFrame object mus have specific values. For example, if the FP_AnchorType is FV_ANCHOR_RUN_INTO_PARAGRAPH, the FP_SideOffset must be 0. The following table lists the constraints on properties for different anchored frame types:

FP_AnchorType	Property constraints
FV_ANCHOR_INLINE	FP_SideOffset = 0
FV_ANCHOR_TOP FV_ANCHOR_BELOW FV_ANCHOR_BOTTOM	FP_SideOffset = 0 FP_BaselineOffset = 0

FP_AnchorType	Property constraints
FV_ANCHOR_RUN_INTO_PARAGRAPH	FP_SideOffset = 0 FP_BaselineOffset = 0 FP_AFrameIsFloating = 0 FP_AFrameIsCropped = 0
FV_ANCHOR_SUBCOL_FARTHEST FV_ANCHOR_SUBCOL_INSIDE FV_ANCHOR_SUBCOL_LEFT FV_ANCHOR_SUBCOL_NEAREST FV_ANCHOR_SUBCOL_OUTSIDE FV_ANCHOR_SUBCOL_RIGHT FV_ANCHOR_TEXTFRAME_FARTHEST FV_ANCHOR_TEXTFRAME_INSIDE FV_ANCHOR_TEXTFRAME_LEFT FV_ANCHOR_TEXTFRAME_NEAREST FV_ANCHOR_TEXTFRAME_OUTSIDE FV_ANCHOR_TEXTFRAME_RIGHT	FP_AFrameIsFloating = 0 FP_AFrameIsCropped = 0

FO_Arc

An FO_Arc object represents an arc. FO_Arc objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845
FP_DTheta	MetricT	Arc angle length (–360 degree to 360 degree)
FP_Theta	MetricT	Start angle (0 degree to 360 degree)

FO_Ellipse

An `FO_Ellipse` object represents an ellipse. `FO_Ellipse` objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845.
`FP_RectangleIsSmoothed`†	`IntT`	`True` if smoothing is enabled. This property is always `True` for `FO_Ellipse` objects.

FO_Group

An `FO_Group` object represents a set of grouped objects. `FO_Group` objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845
`FP_FirstGraphicInGroup`†	`F_ObjHandleT`	First object in the group
`FP_LastGraphicInGroup`†	`F_ObjHandleT`	Last object in the group

FO_Inset

An `FO_Inset` object represents an imported graphic. `FO_Inset` objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845.
Facets		Graphic insets have one or more named properties, called facets. For more information, see “Graphic inset properties” on page 489 of the *FDK Programmer’s Guide*.

Property	Type	Meaning
FP_InsetDpi	IntT	Scaling information for bitmap file (corresponds to the value specified in the Image File Scaling Options dialog box when the graphics file is imported).
FP_ImportHint	StringT	Record identifying the filter used to import the graphic. The FrameMaker product uses this record to find the filter to use when updating the inset. For a complete description of the syntax of this string, see "Syntax of FP_ImportHint strings" on page 855.
FP_InsetEditor	StringT	Name of application to call to edit inset or imported object.
FP_InsetFile	StringT	Platform-specific pathname if the inset is an external inset, or a null string (" ") if it is internal. The pathname can be document-relative. Using F_ApiGetPropVal() or F_ApiGetSring() for FP_InsetFile property for an object returns the disk path for non-HTTP objects and the URL string for HTTP objects. Similarly, using APIs F_ApiSetPropVal() and F_ApiSetSring() will set the original path of the object whether the string is the disk path or a URL.
FP_InsetGfxActiveInPdf	BoolT	If this property is set, on publishing a document to PDF, the inset object that has facets FLV, U3D, or SWF will be activated as soon as the page containing the graphic object is visible. In PDF the graphic objects are called annotation.
FP_InsetGfxName	StringT	Assigns a name to a graphic object. It will work only in case of inset objects that have an FLV, U3D or SWF facet. The name of the graphic should not contain any special characters or spaces.
FP_InsetGfxPlayWindowInPdf	BoolT	If this property is set, on publishing a document to PDF, the inset object that has facets FLV, U3D, or SWF will be activated in a new window in a PDF file. In PDF, the graphic objects are called annotation.
FP_INSETinfo	StringT	Record used to provide general informationa associated with the inset.
FP_InsetIsFixedSize	IntT	True if scaling of bitmap file is inhibited.

Property	Type	Meaning
FP_InsetIsFlipped Sideways	IntT	True if inset is flipped about the vertical axis.
FP_InsetJavaScriptA ttached	BoolT	Denotes whether or not Javascript is attached with the graphic object that has a U3D facet.
FP_InsetJavaScriptF ile	StringT	Attaches the given Javascript file to the graphic object that has a U3D facet. If the value of the file path is null, the Javascript attached to the inset is removed.
FP_InsetMonikerFile Path†	StringT	Provides the file path of the moniker of an inset object that has an OLE2 facet.
FP_InsetMonikerPath †	StringT	Provides the path of the moniker of an inset object that has an OLE2 facet.
FP_InsetPosterFile	IntT	This is used to get and set the path of the poster applied on a graphic object. If the graphic is import by copy, the value is NULL.

FP_InsetSaveFacetTo File		Saves the given facet of an inset to a given file.
		This is set-only property and no get operations are possible.
		The input to the API call contains docid, objId, property name and list of strings. In the list of strings the, first strings is the facet name and the second string is the filename where the facet is to be saved.
		Sample usage:
		`F_StringsT fileArgList;`
		`fileArgList.val = (StringT*)F_Alloc((2) * sizeof(StringT), DSE);`
		`fileArgList.val[0] = F_StrCopyString("JS");`
		`fileArgList.val[1] = F_StrCopyString(afrmObjp ->jsFileName);`
		`fileArgList.len = 2;`
		`F_ApiSetStrings( docId, insetid, FP_InsetSaveFacetToFile, &fileArgList );`

Property	Type	Meaning
FP_InsetU3dAnimationList†	StringT	Provides the list of "animations" defined on the U3D facet of an inset object.
FP_InsetU3dPartList†	StringT	Provides the list of "parts" defined in the U3D facet of an inset object.
FP_InsetU3dViewList†	StringT	Retrieves the list of "views" defined in the U3D facet of an inset object.
FP_InsetUpdater	StringT	Not currently implemented.
FP_InsetURL	StringT	Returns the URL of the inset. Using F_ApiGetPropVal() or F_ApiGetSring() for FP_InsetURL property for an object returns the disk path for non-HTTP objects and the URL string for HTTP objects. Similarly, using APIs F_ApiSetPropVal() and F_ApiSetSring() for will set the original path of the object whether the string is the disk path or a URL.
FP_No3DInPDF	IntT	If True, then U3D files are not embedded in the PDF and ony the poster is embedded in the PDF file created from the FrameMaker file.
FP_NoFlashInPDF	IntT	If True, then Adobe Flash files are not embedded in the PDF and ony the poster is embedded in the PDF file created from the FrameMaker file.

Syntax of FP_ImportHint strings

The FP_ImportHint property of an FO_Inset object specifies a record that identifies the filter used to import a graphic. FrameMaker products use this record to find the correct filter to reimport the graphic when the document is reopened or the inset is manually updated.

The FP_ImportHint property does not apply to graphics imported by copy. FrameMaker products use the facet name stored with the graphic to identify the filter that filtered a graphic imported by copy.

The syntax of the record specified by the FP_ImportHint property is:

record_vers vendor format_id platform filter_vers filter_name

Note that the fields in the record are not separated by spaces. For example:

```
0001PGRFPICTMAC61.0 Built-in PICT reader
```

IMPORTANT: *If you are setting an* `FP_ImportHint` *property, the string you are using should be terminated with* `NULL`*. The string itself must not contain* `NULL` *or undisplayable characters.*

Each field of the record (except *`filter_name`*) specifies a four-byte code. If a code contains fewer than four alphanumeric characters, the remaining bytes must be filled out with spaces.

The rest of this section describes each field in the record.

`record_vers` specifies the version of the record, currently `0001`.

`vendor` is a code specifying the filter's vendor. The code is a string of four characters. The following table lists the possible codes.

Code	Meaning
`PGRF`	Built-in Frame filters
`FAPI`	External Frame FDK client filter
`FFLT`	External Frame filters
`IMAG`	External ImageMark filters
`XTND`	External XTND filters

This is not a comprehensive list of codes. Codes may be added to this list by Frame or by developers at your site.

`format_id` is a code specifying the format that the filter translates. The code is a string of four characters. The following table lists some of the possible codes.

Code	Meaning
`CDR`	CorelDRAW
`CGM`	Computer Graphics Metafile
`DIB`	Device-independent bitmap (Windows)
`DRW`	Micrografx CAD
`DXF`	Autodesk Drawing eXchange file (CAD files)
`EMF`	Enhanced Metafile (Windows)

Code	Meaning
EPSB	Encapsulated PostScript Binary (Windows)
EPSD	Encapsulated PostScript with Desktop Control Separations (DCS)
EPSF	Encapsulated PostScript (Macintosh)
EPSI	Encapsulated PostScript Interchange
FRMI	FrameImage
FRMV	FrameVector
G4IM	CCITT Group 4 to Image
GEM	GEM file (Windows)
GIF	Graphics Interchange Format (Compuserve)
HPGL	Hewlett-Packard Graphics Language
IGES	Initial Graphics Exchange Specification (CAD files)
IMG4	Image to CCITT Group 4 (UNIX)
MooV	QuickTime Movie
OLE	Object Linking and Embedding Client (Microsoft)
PCX	PC Paintbrush
PICT	QuickDraw PICT
PNTG	MacPaint
SNRF	Sun Raster File
SRGB	SGI RGB
TIFF	Tag Image File Format
WMF	Windows Metafile
WPG	WordPerfect Graphics
XWD	X Windows System Window Dump file

platform is a code specifying the platform on which the filter was run.
The code is a string of four characters. The following table lists the possible codes.

Code	Meaning
WINT	Windows NT
WIN3	Windows 3.1
WIN4	Windows 95

filter_vers is a string of four characters identifying the version of the filter on that platform. For example, version 1.0 of a filter is represented by the string `1.0`.

filter_name is a text string (up to 31 characters long) that describes the filter.

FO_Iterator

FO_Iteratgor represents an Iterator object. This allows the user to iterate through the components of the DITAMap or FrameMaker book based on the configuration parameters provided.

FO_KeyCatalog

An `FO_KeyCatalog` object represents the Key Catalogs in FrameMaker. `FO_KeyCatalog` objects have the following properties.

Property	Type	Meaning
`FP_IsDefault`	`BoolT`	If `True`, the key catalog is the default one for the current workflow. If `False`, key catalog is not the default one for the current workflow.
`FP_IsStale`	`BoolT`	If `True`, the key catalog is maked as stale and needs to be re-loaded before using. If `False`, the key catalog is not stale and can be used.
`FP_NextKeyCatalog InSession`	`F_ObjHandleT`	Next key catalog in the session.
`FP_NotLoaded`	`BoolT`	If `True`, the key catalog is not loaded and cannot be used. If `False`, the key catalog is loaded and can be used.
`FP_Source`	`StringT`	Complete path of the file conatining the key catalog.
`FP_SourceType`	`FV_KeySrcTypeNone` `FV_KeySrcTypeDitamap`	Type of the file conatining the key catalog.
`FP_KeyCount`	`IntT`	Number of keys in the key catalog including duplicate definitons.
`FP_KeyCatalogClientName`	`StringT`	Name of the client owning the key catalog.

FO_Line

An `FO_Line` object represents a line. `FO_Line` objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845.
`FP_NumPoints`†	`IntT`	Number of vertices. The default is 2 (the line's start point and end point).
`FP_Points`	`F_PointsT`	Array of x-y coordinate pairs that specify the line's vertices. The default coordinate pairs are for the line's start point and end point.

FO_Math

An `FO_Math` object represents an equation. The `FP_MathFullForm` property corresponds to the MIF `<MathFullForm>` statement that defines the mathematical structure of an equation. Each expression defines a component of the equation and can be nested within other expressions, as in `string1[string2]`.

For example, to create the equation `x > y`, you specify the following string for the `FO_Math` object's `FP_MathFullForm` property:

```
greaterthan[char[x],char[y]]
```

For more information on the MIF `<MathFullForm>` statement, see the online *MIF Reference*.

`FO_Doc` objects have properties that specify how all the equations in a document appear. For a list of these properties, see "Document equation properties" on page 798.

Property	Type	Meaning
Common graphic object properties		See page 845
`FP_BasePointX`	`MetricT`	Horizontal placement of text line base point relative to left side of frame
`FP_BasePointY`	`MetricT`	Vertical placement of text line base point relative to top of frame
`FP_MathFullForm`	`StringT`	String representing the expression

Property	Type	Meaning
FP_MathSize	IntT	Equation size: FV_MATH_LARGE FV_MATH_MEDIUM FV_MATH_SMALL
FP_TextLineType	IntT	Type of text line: FV_TEXTLINE_LEFT FV_TEXTLINE_RIGHT FV_TEXTLINE_CENTER FV_TEXTLINE_MATHD

FO_Polygon

An FO_Polygon object represents a polygon. FO_Polygon objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845
FP_NumPoints	IntT	Number of the polygon's vertices
FP_Points†	F_PointsT	Array of x-y coordinate pairs that specify the polygon's vertices
FP_PolyIsBezier	IntT	True if polygon is smoothed

FO_Polyline

An FO_Polyline object represents a polyline. FO_Polyline objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845
FP_NumPoints	IntT	Number of the polyline's vertices

Property	Type	Meaning
`FP_Points`†	`F_PointsT`	Array of x-y coordinate pairs that specify the polyline's vertices
`FP_PolyIsBezier`	`IntT`	`True` if polyline is a Bezier curve

FO_Rectangle

An `FO_Rectangle` object represents a rectangle. `FO_Rectangle` objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845
`FP_RectangleIsSmoothed`	`IntT`	`True` if smoothing is enabled

FO_RoundRect

An `FO_RoundRect` object represents a rounded rectangle. `FO_RoundRect` objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845
`FP_Radius`	`MetricT`	Radius of corner; `0` for a square corner

FO_TextFrame

An `FO_TextFrame` object represents a text frame. `FO_TextFrame` objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845.
`FP_ColGapWidth`	`MetricT`	Gap between columns (0 to 50 inches).
`FP_ColumnsAreBalanced`	`IntT`	`True` if terminal and underfilled columns in the flow are balanced.
`FP_FirstAFrame`†	`F_ObjHandleT`	First anchored frame in the text frame (`FO_AFrame` ID).
`FP_FirstCell`†	`F_ObjHandleT`	First table cell in the text frame (`FO_Cell` ID).
`FP_FirstFn`†	`F_ObjHandleT`	First footnote in the text frame (`FO_Fn` ID).
`FP_FirstPgf`†	`F_ObjHandleT`	First paragraph in the text frame (`FO_Pgf` ID).
`FP_FirstSubCol`†	`F_ObjHandleT`	First column in the text frame (`FO_SubCol` ID).
`FP_Flow`†	`F_ObjHandleT`	Flow containing the text frame (`FO_Flow` ID).
`FP_GraphicIsButton`	`IntT`	`True` if the text frame is a hypertext button.
`FP_LastAFrame`†	`F_ObjHandleT`	Last anchored frame in the text frame (`FO_AFrame` ID).
`FP_LastCell`†	`F_ObjHandleT`	Last table cell in the text frame (`FO_Cell` ID).
`FP_LastFn`†	`F_ObjHandleT`	Last footnote in the text frame (`FO_Fn` ID).
`FP_LastPgf`†	`F_ObjHandleT`	Last paragraph in the text frame (`FO_Pgf` ID).
`FP_LastSubCol`†	`F_ObjHandleT`	Last column in the text frame (`FO_SubCol` ID).
`FP_NextTextFrameInFlow`	`F_ObjHandleT`	Next text frame in the flow (`FO_TextFrame` ID).

Property	Type	Meaning
FP_NumColumns	IntT	The number of columns in the underlying column grid (1–10).
FP_PrevTextFrameInFlow	F_ObjHandleT	Previous text frame in the flow (FO_TextFrame ID).
FP_SideHeadGap	MetricT	Gap between side head area and body text area (0 to 50 inches).
FP_SideHeadPlacement	IntT	Placement of side heads relative to columns in the text frame: FV_SH_LEFT FV_SH_RIGHT FV_SH_INSIDE FV_SH_OUTSIDE
FP_SideHeadWidth	MetricT	Width of side head area for the text frame (0 to 50 inches).
Text		To get the text in a text frame, call F_ApiGetText() with objId set to the text frame ID. To add text to a text frame, call F_ApiAddText(). To delete text from a text frame, call F_ApiDeleteText(). For more information, see "F_ApiAddText()" on page 80, "F_ApiDeleteText()" on page 151, and "F_ApiGetText()" on page 249.

FO_TextLine

An `FO_TextLine` object represents a text line. `FO_TextLine` objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845.
`FP_BasePointX`	`MetricT`	Horizontal placement of text line base point relative to left side of frame.
`FP_BasePointY`	`MetricT`	Vertical placement of text line base point relative to top of frame.

Property	Type	Meaning
FP_Language	IntT	Obsolete, but still functional. Hyphenation and spell-checking language to use for the first character in the text line. Because language can be specified for a character format, you should get FP_Language from the FO_CharFmt objects in the text of the text line object. The possible values for a text line are: FV_LANG_BRAZILIAN FV_LANG_BRITISH FV_LANG_CANADIAN_FRENCH FV_LANG_CATALAN FV_LANG_DANISH FV_LANG_DUTCH FV_LANG_ENGLISH FV_LANG_FINNISH FV_LANG_FRENCH FV_LANG_GERMAN FV_LANG_ITALIAN FV_LANG_NOLANGUAGE FV_LANG_NORWEGIAN FV_LANG_NYNORSK FV_LANG_PORTUGUESE FV_LANG_SPANISH FV_LANG_SWEDISH FV_LANG_SWISS_GERMAN FV_LANG_JAPANESE FV_LANG_TRADITIONAL_CHINESE FV_LANG_SIMPLIFIED_CHINESE FV_LANG_KOREAN

Property	Type	Meaning
FP_TextLineType	IntT	Type of text line: FV_TEXTLINE_LEFT FV_TEXTLINE_RIGHT FV_TEXTLINE_CENTER FV_TEXTLINE_MATH
Text		To get the text in a text line, call F_ApiGetText() with objId set to the text line ID. To add text to a text line, call F_ApiAddText(). To delete text from a text line, call F_ApiDeleteText(). For more information, see "F_ApiAddText()" on page 80, "F_ApiDeleteText()" on page 151, and "F_ApiGetText()" on page 249.

FO_UnanchoredFrame

An FO_UnanchoredFrame object represents an unanchored frame. FO_UnanchoredFrame objects have the following properties.

Property	Type	Meaning
Common graphic object properties		See page 845.
FP_FirstGraphicInFrame†	F_ObjHandleT	First object in the frame (backmost object).
FP_LastGraphicInFrame†	F_ObjHandleT	Last object in the frame (frontmost object).
FP_Name	StringT	If a reference frame, the frame's name.
FP_PageFramePage†	F_ObjHandleT	If the unanchored frame is a page frame, the page that it belongs to (FO_HiddenPage, FO_BodyPage, FO_MasterPage, or FO_RefPage ID).

FO_Graphicsfmt

The `FO_Graphicsfmt` object is used with the API `F_APInewNamedObject()` to create a new graphics style.
`FO_Graphicsfmt` objects have the following properties.

Property	Type	Meaning
`FP_UseColumnsAreBalanced`	`IntT`	`True` if terminal and underfilled columns in the flow are balanced.

`FP_UseSideHeadPlacement`	`IntT`	Placement of side heads relative to columns in the text frame:
		`FV_SH_LEFT`
		`FV_SH_RIGHT`
		`FV_SH_INSIDE`
		`FV_SH_OUTSIDE`

`FP_UseRadius`	`MetricT`	Radius of corner; `0` for a square corner
`FP_UseFlowIsAutoConnect`	`IntT`	`True` if Autoconnect is enabled.
`FP_UseFlowIsPostScript`	`IntT`	`True` if flow is PostScript code.
`FP_UseBorderWidth`	`MetricT`	Border width (0.015 pt to 360 pt).

`FP_UseFill`	`IntT`	The fill pattern (numbers between 0 and 15; see Figure 3-2). The FDK provides constants for several fill patterns:
		`FV_FILL_BLACK`
		`FV_FILL_WHITE`
		`FV_FILL_CLEAR`

`FP_UsePen`	`IntT`	The pen pattern (numbers between 0 and 15; see Figure 3-2). The FDK provides constants for several pen patterns:
		`FV_FILL_BLACK`
		`FV_FILL_WHITE`
		`FV_FILL_CLEAR`

`FP_UseRunaroundGap`	`MetricT`	If the object is a runaround object, the width of the runaround gap.

Property	Type	Meaning
FP_UseTintPercent	MetricT	The tint percentage
FP_UseOverprint	IntT	Specifies the overprint settings for the object FV_OVERPRINT FV_KNOCKOUT FV_FROMCOLOR
FP_UseAngle	MetricT	Angle of the object's rotation.
FP_UseLocX	MetricT	Distance from the left side of the parent frame
FP_UseLocY	MetricT	Distance from the top side of the parent frame
FP_UseWidth	MetricT	Width of object
FP_UseHeight	MetricT	Height of Object
FP_UseRunaround	MetricT	If the object is a runaround object, the width of the runaround gap.

Property	Type	Meaning
FP_UseAnchorType	IntT	Where frame is anchored: FV_ANCHOR_BELOW FV_ANCHOR_BOTTOM FV_ANCHOR_INLINE FV_ANCHOR_RUN_INTO_PARAGRAPH FV_ANCHOR_SUBCOL_FARTHEST FV_ANCHOR_SUBCOL_INSIDE FV_ANCHOR_SUBCOL_LEFT FV_ANCHOR_SUBCOL_NEAREST FV_ANCHOR_SUBCOL_OUTSIDE FV_ANCHOR_SUBCOL_RIGHT FV_ANCHOR_TEXTFRAME_FARTHEST FV_ANCHOR_TEXTFRAME_INSIDE FV_ANCHOR_TEXTFRAME_LEFT FV_ANCHOR_TEXTFRAME_NEAREST FV_ANCHOR_TEXTFRAME_OUTSIDE FV_ANCHOR_TEXTFRAME_RIGHT FV_ANCHOR_TOP
FP_UseAFrameIsFloating	IntT	True if Floating is enabled
FP_UseAFrameIsCropped	IntT	True if Cropped is enabled
FP_UseSideOffset	MetricT	Near side offset
FP_UseBaselineOffset	MetricT	Baseline offset
FP_UseAlignment	IntT	Type of alignment: FV_ALIGN_CENTER FV_ALIGN_INSIDE FV_ALIGN_OUTSIDE FV_ALIGN_LEFT FV_ALIGN_RIGHT

Property	Type	Meaning
FP_UseNumColumns	IntT	The number of columns in the underlying column grid (1–10).
FP_UseColGapWidth	MetricT	Gap between columns (0 to 50 inches).
FP_UseSideHeadWidth	MetricT	Width of side head area for the text frame (0 to 50 inches).
FP_UseSideHeadGap	MetricT	Gap between side head area and body text area (0 to 50 inches).
FP_UseTheta	MetricT	Start angle (0 degree to 360 degree)
FP_UseDTheta	MetricT	Arc angle length (–360 degree to 360 degree)
FP_UseTextLineType	IntT	Type of text line: FV_TEXTLINE_LEFT FV_TEXTLINE_RIGHT FV_TEXTLINE_CENTER FV_TEXTLINE_MATHD
FP_UseMathSize	IntT	Equation size: FV_MATH_LARGE FV_MATH_MEDIUM FV_MATH_SMALL
FP_UseInsetDpi	IntT	Scaling information for bitmap file (corresponds to the value specified in the Image File Scaling Options dialog box when the graphics file is imported).
FP_FirstGraphicsFmtInDoc	F_ObjHandleT	First graphics format object in the list of the document's graphic format objects
FP_NextGraphicsFmtInDoc	F_ObjHandleT	Next graphic object in the document.
FP_StyleTag	StringT	Name of character format tag applied to text location.)

Insets

For graphic insets, see “FO_Inset” on page 852. For text insets, see “Text insets” on page 937.

Markers

An `FO_Marker` object represents a marker.

FO_Marker

`FO_Marker` objects have the following properties.

Property	Type	Meaning
`FP_Element`†	`F_ObjHandleT`	If the marker is a structured marker in a document, the ID of the element containing the marker.
`FP_MarkerText`	`StringT`	The marker’s text string.
`FP_MarkerTypeId`	`F_ObjHandleT`	The ID of the current marker’s type (`FO_MarkerType`).
`FP_NextMarkerInDoc`†	`F_ObjHandleT`	Next marker (`FO_Marker` ID).
`FP_TextLoc`†	`F_TextLocT`	Text location of the marker’s symbol.
`FP_Unique`†	`IntT`	The marker’s UID.

Marker types

An `FO_MarkerType` object represents a marker type. Marker types are stored in the document.

FO_MarkerType

`FO_MarkerType` objects have the following properties.

Property	Type	Meaning
`FP_NextMarkerTypeInDoc`	`F_ObjHandleT`	Next marker type (`FO_MarkerType` ID)
`FP_Name`	`StringT`	The name of this marker type, as it appears in the user interface.
`FP_InvariantName`	`StringT`	An internal name for the marker type. By default, this is the same as `FP_Name`,. However, this can differ from `FP_Name` if the user interface is in another language.
`FP_OldTypeNum`	`ShortT`	A number to map markers from documents earlier than version 5.5 to this marker type. For example, assume the name of a marker type is `MyMarkerType`, and the `FP_OltTypeNum` is `11`. Then markers of type `11` from earlier documents will import as markers of type `MyMarkerType`.
`FP_Public`	`BoolT`	`True` if the marker type should appear in the user interface. The default is `True`.
`FP_Transient`	`BoolT`	`True` if markers of this type shoud not be saved to files. The default is `False`.
`FP_Required`†	`BoolT`	True if the marker type is required by FrameMaker products. The default is `False`.

Menus

See “Commands, menus, menu items, and menu item separators” on page 760.

Menu items

See “Commands, menus, menu items, and menu item separators” on page 760.

Pages

There are four types of pages. The API uses `FO_BodyPage`, `FO_HiddenPage`, `FO_MasterPage`, and `FO_RefPage` objects to represent them.

FO_BodyPage

An `FO_BodyPage` object represents a body page. `FO_BodyPage` objects have the following properties.

Property	Type	Meaning
`FP_MasterPage`	`StringT`	Name of master page background for body page if `FP_PageBackground` is set to `FV_BGD_OTHER`. It is `NULL` if `FP_PageBackground` is set to `FV_BGD_DEFAULT` or `FV_BGD_NONE`.
`FP_PageBackground`	`IntT`	Type of master page background: `FV_BGD_DEFAULT`: The page has a Left or Right master page background if the document is double-sided, or a Right master page background if the document is single-sided.[a] `FV_BGD_NONE`: The page has no master page background. `FV_BGD_OTHER`: The page has the custom master page background specified by `FP_MasterPage`.
`FP_PageFrame`†	`F_ObjHandleT`	The page’s page frame (`FO_UnanchoredFrame` ID).
`FP_PageHeight`†	`MetricT`	Height of page.
`FP_PageIsRecto`†	`IntT`	`True` if right body page or `False` if left body page.
`FP_PageNext`†	`F_ObjHandleT`	Next body page (`FO_BodyPage` ID) in the document.
`FP_PageNum`†	`IntT`	Page number.

Property	Type	Meaning
FP_PageNumString†	StringT	Page number string.
FP_PagePrev†	F_ObjHandleT	Previous body page (FO_BodyPage ID) in the document.
FP_PageWidth†	MetricT	Width of page.
FP_PointPageNum†	IntT	Number of point page.

a. To determine if a body page has a Left or a Right master page background when its FP_PageBackground property is set to FV_BGD_DEFAULT, query its FP_PageIsRecto property.

FO_HiddenPage

An FO_HiddenPage object represents a hidden page. FO_HiddenPage objects have the following properties.

Property	Type	Meaning
FP_Name†	StringT	Name of hidden page
FP_PageFrame†	F_ObjHandleT	Page frame (FO_UnanchoredFrame ID)
FP_PageHeight†	MetricT	Height of page
FP_PageWidth†	MetricT	Width of page

FO_MasterPage

An FO_MasterPage object represents a master page. FO_MasterPage objects have the following properties.

Property	Type	Meaning
FP_Name	StringT	Name of master page (for example, Right or Left)
FP_PageFrame†	F_ObjHandleT	Page frame (FO_UnanchoredFrame ID)
FP_PageHeight†	MetricT	Height of page
FP_PageNext†	F_ObjHandleT	Next master page (FO_MasterPage ID) in the document

Property	Type	Meaning
FP_PageNum†	IntT	Page number
FP_PagePrev†	F_ObjHandleT	Previous master page (FO_MasterPage ID) in the document
FP_PageWidth†	MetricT	Width of page

FO_RefPage

An FO_RefPage object represents a reference page. FO_RefPage objects have the following properties.

Property	Type	Meaning
FP_Name	StringT	Name of reference page
FP_PageFrame†	F_ObjHandleT	Page frame (FO_UnanchoredFrame ID)
FP_PageHeight†	MetricT	Height of page
FP_PageNext†	F_ObjHandleT	Next reference page (FO_RefPage ID) in the document
FP_PageNum†	IntT	Page number
FP_PagePrev†	F_ObjHandleT	Previous reference page (FO_RefPage ID) in the document
FP_PageWidth†	MetricT	Width of page

Paragraphs

The API uses an FO_Pgf object to represent a paragraph instance and an FO_PgfFmt object to represent a paragraph format in the Paragraph Catalog. FO_Pgf and FO_PgfFmt objects have many of the same properties.

FO_Pgf

FO_Pgf objects have the following properties.

Paragraph Asian character spacing properties

FO_Pgf objects have the following Asian charactrer spacing properties.

Property	Type	Meaning
FP_BkColor	F_ObjHandleT	Text background color
FP_MinJRomSpace	MetricT	Minimum Asian-Roman space
FP_OptJRomSpace	MetricT	Optimum Asian-Roman space
FP_MaxJRomSpace	MetricT	Maximum Asian-Roman space
FP_MinJLetSpace	MetricT	Minimum Asian letter space
FP_OptJLetSpace	MetricT	Optimum Asian letter space
FP_MaxJLetSpace	MetricT	Maximum Asian letter space
FP_YakumonoType	IntT	The Yakumono rules to handle punctuation characters; can be one of FV_FLOATING_YAKUMONO FV_MONOSPACE_YAKUMONO FV_FIXED_YAKUMONO

Paragraph autonumbering properties

FO_Pgf objects have the following autonumbering properties.

Property	Type	Meaning
FP_AutoNumChar	StringT	Character Format for the automatic numbering string specified by FP_AutoNumString; "" if the default character format is used
FP_AutoNumString	StringT	Autonumber Format string; for example, "<n>.<n+>"
FP_NumAtEnd	IntT	True if numbering position is End of Paragraph; False if it is Beginning of Paragraph
FP_PgfIsAutoNum	IntT	True if autonumbering is enabled
FP_PgfNumber†	StringT	The formatted string representation of the paragraph number; for example, 1.2 for a paragraph whose FP_AutoNumString property is set to <n>.<n+>

Paragraph default font properties

FO_Pgf objects have the following default font properties. Most of these properties are the same as text properties.

Property	Type	Meaning
FP_Capitalization	IntT	Type of capitalization to use: FV_CAPITAL_CASE_NORM FV_CAPITAL_CASE_SMALL FV_CAPITAL_CASE_LOWER FV_CAPITAL_CASE_UPPER
FP_ChangeBar	IntT	True if Change Bars are on.
FP_Color	F_ObjHandleT	Spot color (FO_Color ID).
FP_CombinedFontFamily	F_ObjHandleT	Combined font definition (FO_CombinedFontDefn)
FP_FontAngle	IntT	Font angle (specifies an index into the array of font angles provided by the session property FP_FontAngleNames).

Property	Type	Meaning
FP_FontEncodingName†	StringT	The font's encoding
FP_FontFamily	IntT	Font family (specifies an index into the array of font families provided by the session property FP_FontFamilyNames).
FP_FontPlatform Name	StringT	Name that uniquely identifies a font on a specific platform (for more information, see "How the API represents fonts" on page 107 of the *FDK Programmer's Guide*).
FP_FontPostScript Name	StringT	Name given to a font when it is sent to a PostScript printer (for more information, see "How the API represents fonts" on page 107 of the *FDK Programmer's Guide*).
FP_FontSize	MetricT	Font size (2 pt to 400 pt).
FP_FontVariation	IntT	Font variation (specifies an index into the array of font variations provided by the session property FP_FontVariationNames).
FP_FontWeight	IntT	Font weight (specifies an index into the array of font weights provided by the session property FP_FontWeightNames).
FP_KernX	MetricT	Horizontal kern value for manual kerning expressed as a percentage of an em (metric –100% to 1000%).[a] A positive value moves a character right and a negative value moves a character left.
FP_KernY	MetricT	Vertical kern value for manual kerning expressed as a percentage of an em (metric –100% to 1000%). A positive value moves characters up and a negative value moves characters down.
FP_Overline	IntT	True if Overline is enabled.
FP_PairKern	IntT	True if Pair Kern is enabled.

Property	Type	Meaning
FP_Position	IntT	Specifies position relative to baseline of text: FV_POS_NORM: Normal FV_POS_SUB: Subscript FV_POS_SUPER: Superscript
FP_Spread	MetricT	Obsolete property, but still functional. See corresponding "tracking" property below.
FP_Stretch	MetricT	Character stretch (set width) expressed as a percentage of normal stretch for the font (metric –10% to 1000%).
FP_Strikethrough	IntT	True if Strikethrough is enabled.
FP_Tracking	MetricT	Character tracking expressed as a percentage of an em (metric –100% to 1000%).
FP_Underlining	IntT	Type of underlining: FV_CB_NO_UNDERLINE FV_CB_SINGLE_UNDERLINE FV_CB_DOUBLE_UNDERLINE FV_CB_NUMERIC_UNDERLINE

a. In the API, most percentages are represented as MetricT fractions. For spread percentages, the MetricT value 1<<16 or 0x10000 specifies 100% or 1. For more information on MetricT values, see “MetricT values” on page 954.

Paragraph formats in a table cell

FO_Pgf objects have the following properties that specify how a paragraph appears in a table cell.

Property	Type	Meaning
FP_CellBottomMargin	MetricT	Amount added to default bottom margin of table cell.
FP_CellLeftMargin	MetricT	Amount added to default left margin of table cell.
FP_CellMarginsFixed	IntT	Specifies which cell margins are fixed. The following values can be ORed into it: FV_PGF_FIXED_B_MARGIN: the bottom margin is fixed FV_PGF_FIXED_L_MARGIN: the left margin is fixed FV_PGF_FIXED_R_MARGIN: the right margin is fixed FV_PGF_FIXED_T_MARGIN: the top margin is fixed If the margin for a cell is fixed, the margin property specifies the absolute value of the cell margin. For example, if FV_PGF_FIXED_B_MARGIN is set, FP_CellBottomMargin specifies the absolute value of the cell's bottom margin, overriding the cell margin specified by the table format. If FV_PGF_FIXED_B_MARGIN is not set, FP_CellBottomMargin is added to the margin specified by the table format.
FP_CellRightMargin	MetricT	Amount added to default right margin of table cell.
FP_CellTopMargin	MetricT	Amount added to default top margin of table cell.
FP_CellVAlignment	IntT	Vertical alignment of a paragraph when it is the first one in a cell: FV_PGF_V_ALIGN_TOP FV_PGF_V_ALIGN_MIDDLE FV_PGF_V_ALIGN_BOTTOM

Paragraph general properties

`FO_Pgf` objects have the following general properties.

Property	Type	Meaning
`FP_Locked`	`IntT`	`True` if the paragraph is part of a text inset that retains formatting information from the source document. The paragraph is not affected by global formatting performed on the document.
`FP_FormatOverride`	`IntT`	`True` if the paragraph contains a paragraph format override.

Paragraph hyphenation properties

`FO_Pgf` objects have the following hyphenation properties.

Property	Type	Meaning
`FP_AdjHyphens`	`IntT`	Number of allowable adjacent hyphens
`FP_Hyphenate`	`IntT`	`True` if Automatic Hyphenation is enabled
`FP_HyphMinPrefix`	`IntT`	Minimum number of letters that must precede hyphen
`FP_HyphMinSuffix`	`IntT`	Minimum number of letters that must follow a hyphen
`FP_HyphMinWord`	`IntT`	Minimum length of a hyphenated word

Paragraph hyphenation and spell-checking languages

`FO_Pgf` objects have the following language properties.

Property	Type	Meaning

`FP_Language`	`IntT`	Hyphenation and spell-checking language to use:[a]
		`FV_LANG_BRAZILIAN`
		`FV_LANG_BRITISH`
		`FV_LANG_CANADIAN_FRENCH`
		`FV_LANG_CATALAN`
		`FV_LANG_DANISH`
		`FV_LANG_DUTCH`
		`FV_LANG_ENGLISH`
		`FV_LANG_FINNISH`
		`FV_LANG_FRENCH`
		`FV_LANG_GERMAN`
		`FV_LANG_ITALIAN`
		`FV_LANG_NOLANGUAGE`
		`FV_LANG_NORWEGIAN`
		`FV_LANG_NYNORSK`
		`FV_LANG_PORTUGUESE`
		`FV_LANG_SPANISH`
		`FV_LANG_SWEDISH`
		`FV_LANG_SWISS_GERMAN`
		`FV_LANG_JAPANESE`
		`FV_LANG_TRADITIONAL_CHINESE`
		`FV_LANG_SIMPLIFIED_CHINESE`
		`FV_LANG_KOREAN`

`FP_PgfSpellChecked`	`IntT`	`True` if paragraph has been spell-checked

a. The FDE provides a function that returns the language string associated with each of the language constants. For more information, see "F_LanguageString()" on page 576.

Paragraph identifiers

`FO_Pgf` objects have the following identification property.

Property	Type	Meaning
`FP_Unique`†	`IntT`	The paragraph's UID

Paragraph indents

`FO_Pgf` objects have the following indentation properties.

Property	Type	Meaning
`FP_FirstIndent`	`MetricT`	First-line left margin, measured from left side of current text column (0 cm to 100 cm)
`FP_LeftIndent`	`MetricT`	Left margin, measured from left side of current text column (0 cm to 100 cm)
`FP_RightIndent`	`MetricT`	Right margin, measured from right side of current text column

Paragraph line spacing

`FO_Pgf` objects have the following line-spacing properties.

Property	Type	Meaning
`FP_Leading`	`MetricT`	Space below each line in a paragraph
`FP_LineSpacing`	`IntT`	Space between lines in a paragraph measured from baseline to baseline: `FV_PGF_FIXED`: default font size `FV_PGF_PROPORTIONAL`: largest font in line `FV_PGF_FLOATING`: largest ascender in line

Paragraph placement properties

FO_Pgf objects have the following placement properties.

Property	Type	Meaning
FP_BlockLines	IntT	The number of Widow/Orphan lines
FP_KeepWithNext	IntT	True if Keep With Next Paragraph is enabled
FP_KeepWithPrev	IntT	True if Keep With Previous Paragraph is enabled
FP_PgfAlignment	IntT	Horizontal alignment of paragraph: FV_PGF_LEFT FV_PGF_RIGHT FV_PGF_CENTER FV_PGF_JUSTIFIED
FP_Placement	IntT	Paragraph placement: FV_PGF_SIDEBODY FV_PGF_SIDEHEAD_TOP FV_PGF_SIDEHEAD_FIRST_BASELINE FV_PGF_SIDEHEAD_LAST_BASELINE FV_PGF_RUN_IN FV_PGF_STRADDLE FV_PGF_STRADDLE_NORMAL_ONLY
FP_RunInSeparator	StringT	String for Run-In Head Default Punctuation
FP_SpaceAbove	MetricT	Space above paragraph
FP_SpaceBelow	MetricT	Space below paragraph
FP_Start	IntT	Vertical placement of paragraph: FV_PGF_ANYWHERE FV_PGF_TOP_OF_COL FV_PGF_TOP_OF_PAGE FV_PGF_TOP_OF_LEFT_PAGE FV_PGF_TOP_OF_RIGHT_PAGE

Paragraph object pointer properties

`FO_Pgf` objects have the following properties that specify the IDs of other objects.

Property	Type	Meaning
`FP_InTextFrame`†	`F_ObjHandleT`	Text frame containing the paragraph (`FO_TextFrame` ID)
`FP_InTextObj`†	`F_ObjHandleT`	Subcolumn, footnote, or table cell the paragraph begins in (`FO_SubCol`, `FO_Fn`, or `FO_Cell` ID)
`FP_NextPgfInDoc`†	`F_ObjHandleT`	Next paragraph in the document (`FO_Pgf` ID)
`FP_NextPgfInFlow`†	`F_ObjHandleT`	Next paragraph in the flow (`FO_Pgf` ID)
`FP_PrevPgfInFlow`†	`F_ObjHandleT`	Previous paragraph in the flow (`FO_Pgf` ID)

Paragraph reference frame

`FO_Pgf` has the following properties that specify reference frames.

Property	Type	Meaning
`FP_BottomSeparator`	`StringT`	Name of frame to put below paragraph
`FP_TopSeparator`	`StringT`	Name of frame to put above paragraph

Paragraph tabs

`FO_Pgf` objects have the following tab properties.

Property	Type	Meaning
`FP_NumTabs`†	`IntT`	Number of tabs in the paragraph.
`FP_Tabs`	`F_TabsT`	Array of tab descriptions that specify the positions and types of tab stops. For an example of how to get and set tabs in a paragraph, see “F_ApiSetTabs()” on page 441.

Paragraph tagging

FO_Pgf objects have the following format tag properties.

Property	Type	Meaning
FP_Name	StringT	Name of paragraph format
FP_NextTag	StringT	Tag for new next paragraph
FP_UseNextTag	IntT	True if Next Paragraph Tag is enabled

Paragraph text

The API does not use properties to represent the text in a paragraph. To get the text in a paragraph, call F_ApiGetText() with objId set to the paragraph's ID. To add text to a paragraph, call F_ApiAddText(). To delete text from a paragraph, call F_ApiDeleteText(). For more information, see "F_ApiAddText()" on page 80, "F_ApiDeleteText()" on page 151, and "F_ApiGetText()" on page 249.

Paragraph word spacing

FO_Pgf objects have the following word spacing properties.

Property	Type	Meaning
FP_LetterSpace	IntT	True if Word Spacing is enabled
FP_MaxSpace	MetricT	Maximum word spacing (percentage of an em space in current font)
FP_MinSpace	MetricT	Minimum word spacing (percentage of an em space in current font)
FP_OptSpace	MetricT	Optimum word spacing

FO_PgfFmt

FO_PgfFmt objects (paragraph formats in the Paragraph Catalog) have the following properties.

Paragraph format PDF properties

FO_PgfFmt objects have the following PDF properties.

Property	Type	Meaning
FP_AcrobatLevel	IntT	Retained in Version 6.0 or later for backward compatibility. Use FP_PDFLevel instead.
FP_BkColor	F_ObjHandleT	Text background color
FP_MarkedForNamedDestination	IntT	If True, this paragraph will have a corresponding Named Destination in the generated PDF.
FP_PDFLevel	IntT	The outline level of paragraphs with this format when the FrameMaker product generates PDF data. The value for this property can be between 0 and 100, where greater values are deeper in the hierarchy. If FP_AcrobatLevel is 0, the FrameMaker product does not generate bookmarks for paragraphs with the format. If FP_AcrobatLevel is greater than 0, the FrameMaker product sets the outline level of paragraphs with this format to the specified level.
FP_PDFStructureLevel	IntT	The PDF structure level of paragraphs with the current format. This property is used when the FP_PDFStructure property is True for the document, and the FrameMaker product generates PDF data. The value for this property can be between 0 and 100, where greater values are deeper in the hierarchy. If FP_PDFStructureLevel is 0, the FrameMaker product does not include paragraphs of this format in the PDF structure.

Paragraph format Asian character spacing properties

FO_Pgf objects have the following Asian character spacing properties.

Property	Type	Meaning
FP_MinJRomSpace	MetricT	Minimum Asian-Roman space
FP_OptJRomSpace	MetricT	Optimum Asian-Roman space
FP_MaxJRomSpace	MetricT	Maximum Asian-Roman space

Property	Type	Meaning
FP_MinJLetSpace	MetricT	Minimum Asian letter space
FP_OptJLetSpace	MetricT	Optimum Asian letter space
FP_MaxJLetSpace	MetricT	Maximum Asian letter space
FP_YakumonoType	IntT	The Yakumono rules to handle punctuation characters; can be one of FV_FLOATING_YAKUMONO FV_MONOSPACE_YAKUMONO FV_FIXED_YAKUMONO

Paragraph format autonumbering properties

FO_PgfFmt objects have the following autonumbering properties.

Property	Type	Meaning
FP_AutoNumChar	StringT	Character format for the automatic numbering string specified by FP_AutoNumString; "" if the default character format is used
FP_AutoNumString	StringT	Autonumber format string (for example, <n>.<n+>)
FP_NumAtEnd	IntT	True if numbering position is End of Paragraph; False if it is Beginning of Paragraph
FP_PgfIsAutoNum	IntT	True if autonumbering is enabled

Paragraph format default font properties

FO_PgfFmt objects have the following default font properties.

Property	Type	Meaning
FP_Capitalization	IntT	Type of capitalization to use: FV_CAPITAL_CASE_NORM FV_CAPITAL_CASE_SMALL FV_CAPITAL_CASE_LOWER FV_CAPITAL_CASE_UPPER
FP_ChangeBar	IntT	True if Change Bars is on.

Property	Type	Meaning
FP_Color	F_ObjHandleT	Spot color (FO_Color ID).
FP_CombinedFontFamily	F_ObjHandleT	Combined font definition (FO_CombinedFontDefn)
FP_FontAngle	IntT	Font angle (specifies an index into the array of font angles provided by the session property FP_FontAngleNames).
FP_FontEncodingName†	StringT	The font's encoding
FP_FontFamily	IntT	Font family (specifies an index into the array of font families provided by the session property FP_FontFamilyNames).
FP_FontPlatformName	StringT	Name that uniquely identifies a font on a specific platform (for more information, see "How the API represents fonts" on page 107 of the *FDK Programmer's Guide*).
FP_FontPostScriptName	StringT	Name given to a font when it is sent to a PostScript printer (for more information, see "How the API represents fonts" on page 107 of the *FDK Programmer's Guide*).
FP_FontSize	MetricT	Font size (2 pt to 400 pt).
FP_FontVariation	IntT	Font variation (specifies an index into the array of font variations provided by the session property FP_FontVariationNames).
FP_FontWeight	IntT	Font weight (specifies an index into the array of font weights provided by the session property FP_FontWeightNames).
FP_KernX	MetricT	Horizontal kern value for manual kerning expressed as a percentage of an em (metric –100% to 1000%).[a] A positive value moves a character right and a negative value moves a character left.

Property	Type	Meaning
FP_KernY	MetricT	Vertical kern value for manual kerning expressed as a percentage of an em (metric –100% to 1000%). A positive value moves characters up and a negative value moves characters down.
FP_Overline	IntT	True if Overline is enabled.
FP_PairKern	IntT	True if Pair Kern is enabled.
FP_Position	IntT	Specifies position relative to baseline of text: FV_POS_NORM: Normal FV_POS_SUB: Subscript FV_POS_SUPER: Superscript
FP_Spread	MetricT	Character spread expressed as a percentage of an em (metric –100% to 1000%).
FP_Stretch	MetricT	Character stretch (set width) expressed as a percentage of normal stretch for the font (metric –10% to 1000%).
FP_Strikethrough	IntT	True if Strikethrough is enabled.
FP_Tracking	MetricT	Character tracking expressed as a percentage of an em (metric –100% to 1000%).
FP_Underlining	IntT	Type of underlining: FV_CB_NO_UNDERLINE FV_CB_SINGLE_UNDERLINE FV_CB_DOUBLE_UNDERLINE FV_CB_NUMERIC_UNDERLINE

a. In the API, most percentages are represented as MetricT fractions. For spread percentages, the MetricT value 1<<16 or 0x10000 specifies 100% or 1. For more information on MetricT values, see "MetricT values" on page 954.

Paragraph format hyphenation properties

`FO_PgfFmt` objects have the following properties that specify how words are hyphenated.

Property	Type	Meaning
`FP_AdjHyphens`	`IntT`	Number of allowable adjacent hyphens
`FP_Hyphenate`	`IntT`	`True` if Hyphenation is enabled
`FP_HyphMinPrefix`	`IntT`	Minimum number of letters that must precede hyphen
`FP_HyphMinSuffix`	`IntT`	Minimum number of letters that must follow a hyphen
`FP_HyphMinWord`	`IntT`	Minimum length of a hyphenated word

Paragraph format spell-checking languages

`FO_PgfFmt` objects have the following language properties. The FDE provides a function that returns the language string associated with each of the language constants. For more information, see "F_LanguageString()" on page 576.

Property	Type	Meaning

`FP_Language`	`IntT`	Hyphenation and spell-checking language to use:
		`FV_LANG_BRAZILIAN`
		`FV_LANG_BRITISH`
		`FV_LANG_CANADIAN_FRENCH`
		`FV_LANG_CATALAN`
		`FV_LANG_DANISH`
		`FV_LANG_DUTCH`
		`FV_LANG_ENGLISH`
		`FV_LANG_FINNISH`
		`FV_LANG_FRENCH`
		`FV_LANG_GERMAN`
		`FV_LANG_ITALIAN`
		`FV_LANG_NOLANGUAGE`
		`FV_LANG_NORWEGIAN`
		`FV_LANG_NYNORSK`
		`FV_LANG_PORTUGUESE`
		`FV_LANG_SPANISH`
		`FV_LANG_SWEDISH`
		`FV_LANG_SWISS_GERMAN`
		`FV_LANG_JAPANESE`
		`FV_LANG_TRADITIONAL_CHINESE`
		`FV_LANG_SIMPLIFIED_CHINESE`
		`FV_LANG_KOREAN`

Paragraph format indents

FO_PgfFmt objects have the following indentation properties.

Property	Type	Meaning
FP_FirstIndent	MetricT	First-line left margin, measured from left side of current text column (0 pt to 360 pt)
FP_LeftIndent	MetricT	Left margin, measured from left side of current text column (0 pt to 360 pt)
FP_RightIndent	MetricT	Right margin, measured from right side of current text column

Paragraph format line spacing

FO_PgfFmt objects have the following line-spacing properties.

Property	Type	Meaning
FP_Leading	MetricT	Space below each line in a paragraph
FP_LineSpacing	IntT	Space between lines in a paragraph measured from baseline to baseline: FV_PGF_FIXED: default font size FV_PGF_PROPORTIONAL: largest font in line FV_PGF_FLOATING: largest ascender in line

Paragraph format object pointer properties

FO_PgfFmt objects have the following property that specifies the ID of another FO_PgfFmt object.

Property	Type	Meaning
FP_NextPgfFmtInDoc†	F_ObjHandleT	Next paragraph format in document (FO_PgfFmt ID)

Paragraph format placement properties

`FO_PgfFmt` objects have the following placement properties.

Property	Type	Meaning
`FP_BlockLines`	`IntT`	The number of Widow/Orphan lines
`FP_KeepWithNext`	`IntT`	`True` if Keep With Next Paragraph is enabled
`FP_KeepWithPrev`	`IntT`	`True` if Keep With Previous Paragraph is enabled

`FP_PgfAlignment`	`IntT`	Horizontal alignment of paragraph:
		`FV_PGF_LEFT`
		`FV_PGF_RIGHT`
		`FV_PGF_CENTER`
		`FV_PGF_JUSTIFIED`

`FP_Placement`	`IntT`	Paragraph placement:
		`FV_PGF_SIDEBODY`
		`FV_PGF_SIDEHEAD_TOP`
		`FV_PGF_SIDEHEAD_FIRST_BASELINE`
		`FV_PGF_SIDEHEAD_LAST_BASELINE`
		`FV_PGF_RUN_IN`
		`FV_PGF_STRADDLE`
		`FV_PGF_STRADDLE_NORMAL_ONLY`

`FP_RunInSeparator`	`StringT`	String for Run-In Head Default Punctuation
`FP_SpaceAbove`	`MetricT`	Space before paragraph
`FP_SpaceBelow`	`MetricT`	Space after paragraph

`FP_Start`	`IntT`	Vertical placement of paragraph:
		`FV_PGF_ANYWHERE`
		`FV_PGF_TOP_OF_COL`
		`FV_PGF_TOP_OF_PAGE`
		`FV_PGF_TOP_OF_LEFT_PAGE`
		`FV_PGF_TOP_OF_RIGHT_PAGE`

Paragraph format reference frame properties

`FO_PgfFmt` objects have the following reference frame properties.

Property	Type	Meaning
`FP_BottomSeparator`	`StringT`	Name of frame to put below paragraph
`FP_TopSeparator`	`StringT`	Name of frame to put above paragraph

Paragraph format table cell properties

`FO_PgfFmt` objects have the following properties that specify how a paragraph appears in a table cell.

Property	Type	Meaning
`FP_CellBottomMargin`	`MetricT`	Amount added to table cell default bottom margin
`FP_CellLeftMargin`	`MetricT`	Amount added to table cell default left margin
`FP_CellMarginsFixed`	`IntT`	Specifies which cell margins are added to default table cell margins. The following values can be ORed into it. `FV_PGF_FIXED_B_MARGIN`: the bottom margin is added `FV_PGF_FIXED_L_MARGIN`: the left margin is added `FV_PGF_FIXED_R_MARGIN`: the right margin is added `FV_PGF_FIXED_T_MARGIN`: the top margin is added
`FP_CellRightMargin`	`MetricT`	Amount added to table cell default right margin
`FP_CellTopMargin`	`MetricT`	Amount added to table cell default top margin

`FP_CellVAlignment`	`IntT`	Vertical alignment of a paragraph when it is the first one in a cell:
		`FV_PGF_V_ALIGN_TOP`
		`FV_PGF_V_ALIGN_MIDDLE`
		`FV_PGF_V_ALIGN_BOTTOM`

Paragraph format tabs

FO_PgfFmt objects have the following format tab properties.

Property	Type	Meaning
FP_NumTabs†	IntT	Number of tabs in the paragraph
FP_Tabs	F_TabsT	Array of tab descriptions that specify the positions and types of tab stops

Paragraph format tagging

FO_PgfFmt objects have the following format tag properties.

Property	Type	Meaning
FP_Name	StringT	Name of paragraph format
FP_NextTag	StringT	Tag for following paragraph
FP_UseNextTag	IntT	True if Next Paragraph Tag is enabled

Paragraph format word spacing

FO_PgfFmt objects have the following word-spacing properties.

Property	Type	Meaning
FP_LetterSpace	IntT	True if Word Spacing is enabled
FP_MaxSpace	MetricT	Maximum word spacing (percentage of an em space in current font)[a]
FP_MinSpace	MetricT	Minimum word spacing (percentage of an em space in current font)
FP_OptSpace	MetricT	Optimum word spacing

a. In the API, most percentages are represented as MetricT fractions. For word spacing percentages, the MetricT value 1<<16 or 0x10000 specifies 100% or one em space. For more information on MetricT values, see “MetricT values” on page 954.

Rubi composites

An `FO_Rubi` object represents a rubi composite.

FO_Rubi

`FO_Rubi` objects have the following properties.

Property	Type	Meaning
`FP_Element`†	`F_ObjHandleT`	If the rubi group is in a structured document, the object handle of the associated FO_Element for the rubi group element.
`FP_OyamojiTextRange`†	`F_TextRangeT`	The text range the oyamoji text encompasses.
`FP_NextRubiInDoc`†	`F_TextRangeT`	The next instance of a rubi composite (`FO_Rubi` ID) in the document..
`FP_RubiElement`†	`F_ObjHandleT`	If the rubi group is in a structured document, the object handle of the associated FO_Element for the rubi element.
`FP_RubiTextRange`†	`F_TextRangeT`	The text range the rubi text encompasses.
`FP_Unique`†	`IntT`	The rubi composite's UID.

Table ruling formats

The API uses an `FO_RulingFmt` object to represent each ruling format in a document. Tables and table formats specify rulings by specifying the IDs of `FO_RulingFmt` objects.

FO_RulingFmt

`FO_RulingFmt` objects have the following properties.

Property	Type	Meaning
`FP_Name`	`StringT`	Ruling format name.
`FP_NextRulingFmtInDoc`†	`F_ObjHandleT`	Next ruling format in document (`FO_RulingFmt` ID).

`FP_Pen`	`IntT`	The pen pattern (numbers between 0 and 15; see Figure 3-2). The FDK provides constants for several pen patterns:
		`FV_FILL_BLACK`
		`FV_FILL_WHITE`
		`FV_FILL_CLEAR`

`FP_RulingGap`	`MetricT`	Gap between double ruling lines (0.015 pt to 360 pt).
`FP_RulingLines`	`IntT`	Number of ruling lines (0 to 2 lines).
`FP_RulingPenWidth`	`MetricT`	Ruling line thickness (0.015 pt to 360 pt).
`FP_RulingSep`	`F_ObjHandleT`	Spot color of ruling format (`FO_Color` ID).

Separators

See “Commands, menus, menu items, and menu item separators” on page 760.

Session

The API uses an `FO_Session` object to represent a FrameMaker product session.

FO_Session

`FO_Session` objects have the following properties.

Property	Type	Meaning
`FP_ActiveBook`	`F_ObjHandleT`	The book with input focus (`FO_Book` ID).
`FP_ActiveDoc`	`F_ObjHandleT`	The document with input focus (`FO_Doc` ID).

`FP_ActiveView`	`StringT`	Sets the current view. The view can be one of:
		• `WYSIWYG View`
		• `Author View`
		• `XML View`
		To get the current view:
		`StringT activeView = F_ApiGetString (FV_SessionId, FV_SessionId, FP_ActiveView);`
		To set the current view:
		`StringT viewName = "WYSIWYG View"`
		`F_ApiSetString (FV_SessionId, FV_SessionId, FP_ActiveView, viewName);`

`FP_AllowNewFileURL`	`IntT`	If True, allows file URLs starting with "file:" as well. Otherwise, only file URLs starting with file:// are allowed.
`FP_ApplyFormatRules`	`IntT`	`True` if element reformatting is enabled (Structured FrameMaker only).
`FP_AddMarkerTypeTo StandardMarkers`	`StringT`	The name of a marker type to add to the standard list of marker types. Use `F_ApiSetString()` to set a marker type name to this property of the `FV_SessionId`. See "The standard list of marker types" on page 125 of the *FDK Programmer's Guide*.

Property	Type	Meaning
FP_AutoBackup	IntT	True if Automatic Backup is enabled.
FP_AutoSave	IntT	True if Automatic Save is enabled.
FP_AutoSaveSeconds	IntT	Time between automatic saves in seconds (60 seconds to 10800 seconds).
FP_DefaultVectorFormatForXMLExport	StringT	Used to set/get the default format (such as CGM or MIF) that is used for exporting the Anchored Frame or Equation while exporting FrameMaker document as XML. For example: `F_ApiSetString(0, FV_SessionId, FP_DefaultVectorFormatForXMLExport, "CGM");`
FP_Displaying	IntT	False if screen refresh is completely turned off.
FP_DoNotExportInvalidXML	IntT	If True, XML is not exported/saved if it is invalid.
FP_DoPostXSLTValidationOnExport	IntT	If True, XML is validated after doing XSLT while exporting.
FP_EnableAutoSpellCheck	IntT	If True, then enable auto spell check.
FP_FirstCommandInSession	F_ObjHandleT	First command in the list of commands in the session (FO_Command ID).
FP_FirstMenuItemInSession†	F_ObjHandleT	First menu item or menu in the list of menus, menu items, and menu item separators in the session (FO_Command, FO_Menu, FO_MenuItemSeparator ID).
FP_FirstOpenBook†	F_ObjHandleT	First open book (FO_Book ID) in session.
FP_FirstOpenDoc†	F_ObjHandleT	First open document (FO_Doc ID) in session.
FP_FM_BinDir†	StringT	Directory pathname of $FMHOME/bin
FP_FM_CurrentDir†	StringT	Name of the directory from which the FrameMaker product was started

Property	Type	Meaning
FP_FM_StructureDir†	StringT	Directory pathname of $FMHOME/structure.
FP_CurrentMenuSet	IntT	Type of menu set: FV_MENU_QUICK FV_MENU_COMPLETE FV_MENU_CUSTOM
FP_DefaultKeyCatalog	F_ObjHandleT	Default key catalog for the current workflow.
FP_ExportFilters	StringListT	List of export filters available in the current session.
FP_FirstKeyCatalogInSession	F_ObjHandleT	First key catalog in the session.

FP_FMConsoleString	StringT	Get, append, or clear the FrameMaker console string. Here are examples of usage:
		Get:
		`StringT str = F_ApiGetString (FV_SessionId, FV_SessionId, FP_FMConsoleString);`
		Append:
		`F_ApiSetString (FV_SessionId, FV_SessionId, FP_FMConsoleString, (StringT)"another line in console");`
		Clear:
		`F_ApiSetString (FV_SessionId, FV_SessionId, FP_FMConsoleString, (StringT)"-1");`

FP_FM_HelpDir†	StringT	Pathname of the FrameMaker product help directory
FP_FM_HomeDir†	StringT	Pathname of $FMHOME .
FP_FM_InitDir†	StringT	Directory pathname of $FMHOME/fminit .

Property	Type	Meaning
FP_FontAngleNames†	F_StringsT	List of font angles available in the current session.
FP_FontFamily Attributes†	F_IntsT	An array of flags that indicate attributes for each font family listed by FP_FontFamilyNames. This array of integers is indexed the same as the list of font family names, and corresponds directly to that list. Each IntT is a packed field; the high order 16 bits indicate a surrogate font, and the low order bits indicate attributes for the font family. The flags, their mask values, and their meaning follow: FV_FAMILY_VISIBLE (0x00000001): Family is visible in menu. FV_FAMILY_SELECTABLE (0x00000002): Family can be selected in menu. FV_FAMILY_MAPPED (0x00000004): Family is is always mapped to another family. FV_FAMILY_SURROGATE (0xFFFF0000): The family mapped to if FV_FAMILY_MAPPED is True.
FP_FontFamilyNames†	F_StringsT	List of font family names available in the current session. Note that this list does not include combined fonts (see "FO_CombinedFontDefn" on page 759)
FP_FontVariation Names†	F_StringsT	List of font variations available in the current session.
FP_FontWeightNames†	F_StringsT	List of font weights available in the current session.
FP_Gravity	IntT	True if Gravity is turned on for the session.
FP_GreekSize	MetricT	Size at which to greek text.
FP_HostName†	StringT	Name of the host computer.

Property	Type	Meaning
FP_IconBarOn	IntT	True if the four icons that appear on the upper-right side of the document window are on. Changing this property affects only documents that are opened subsequently; it does not affect documents that are already open.
FP_ImportFilters	StringListT	List of import filters available in the current session.
FP_IsFMRunningAsServer	IntT	True if this instance of FrameMaker is FrameMaker Server.
FP_IsIconified	IntT	True if the FrameMaker product window is iconified.
FP_IsInFront	IntT	True if the FrameMaker product window is in front of other application windows. You can use this property to bring the FrameMaker product to the front or back.
FP_IsTempOpenSave	IntT	Gets whether temporary open/save is in progress. Temporary open/save happens during view switching operations. Here is an example of getting this value: `IntT isTempOpenSave = F_ApiGetInt (FV_SessionId, FV_SessionId, FP_IsTempOpenSave);`

FP_KeyCatalogWorkflow	IntT	Current workflow related to key catalogs. The possible values are:
		`FV_KeyCatalogWorkflowAuthoring`
		`FV_KeyCatalogWorkflowPublishing`
		`FV_KeyCatalogWorkflowSearching`

FP_Label	StringT	The title in the FrameMaker product window title bar.

Property	Type	Meaning
FP_Language†	IntT	Product language: FV_LANG_BRITISH FV_LANG_ENGLISH FV_LANG_FRENCH FV_LANG_GERMAN FV_LANG_ITALIAN FV_LANG_NOLANGUAGE FV_LANG_SPANISH FV_LANG_SWEDISH FV_LANG_JAPANESE FV_LANG_TRADITIONAL_CHINESE FV_LANG_SIMPLIFIED_CHINESE FV_LANG_KOREAN
FP_MarkerNames†	F_StringsT	List of standard marker types for the current session. For versions prior to 5.5, this returned the list of all marker types for the current session. In version 5.5, marker types are assigned to the document; use the FP_MarkerTypeNames property of FO_Doc to get the full list of marker types.
FP_NoMultiMediaInPDF	IntT	If True, multimedia is not embedded when creating PDF from FM.
FP_OpenDir	StringT	Directory pathname of the FrameMaker product directory.
FP_OperatingSystem†	StringT	Operating system under which the current session is running: DOS
FP_Path†	StringT	Pathname to search to start the FrameMaker product.
FP_Platform†	StringT	Name of the platform on which the current session is running: Intel
FP_ProductIsDemo	BoolT	True if the current session is for a demo version of FrameMaker

Property	Type	Meaning
FP_ProductIsStructured	BoolT	True if FrameMaker is running in structured mode for the current session
FP_ProductName†	StringT	FrameMaker product name: The names for FrameMaker+SGML indicate FrameMaker running under the structured product interface. FrameViewer is retained for backward compatibility. Can be one of: FrameMaker FrameMaker+SGML FrameViewer DemoMaker DemoMaker+SGML
FP_ProgId	StringT	Returns the Program ID of the current FrameMaker session
FP_Reformatting	IntT	True if reformatting is enabled.
FP_RememberMissing FontNames	IntT	True if Remember Missing Font Names is activated .
FP_RemoveExtraWhiteSpa cesOnXMLImport	IntT	If True, removes extra whitespace from the XML while importing.
FP_ScreenHeight	IntT	Height of the FrameMaker product window in pixels.
FP_ScreenWidth	IntT	Width of the FrameMaker product window in pixels.
FP_ScreenX	IntT	The offset of the FrameMaker product window in pixels from the left side of the screen. If you set a value that would result in the product window being off the screen, that value is ignored and the old value is retained.
FP_ScreenY	IntT	The offset of the FrameMaker product window in pixels from the top of the screen. If you set a value that would result in the product window being off the screen, that value is ignored and the old value is retained.

Property	Type	Meaning
FP_Snap	IntT	True if Snap is turned on for the session.
FP_TrackChangesOn	Boolean	Determines whether track changes in On or Off for a document
FP_StackWarningLevel	IntT	Determines how warnings are displayed when history-clearing operations occur. It corresponds to an option set in the Preferences dialog, and to the preference-file flag hpWarning. Use F_ApiSetInt to set this property value, and F_ApiGetInt to retrieve it. Allowed values are: – FvWarnNever: Disables warnings for history-clearing operations for the session. – FvWarnOnce: Displays a warning when a particular history-clearing command is issued,but does not warn on subsequent uses of that command. – FvWarnAlways: Displays warnings every time a history-clearing command is issued.
FP_StructAppAttrConfig File	StringT	Set an attribute config file for a structured application.
FP_TmpDir†	StringT	Pathname of temporary directory for internal FrameMaker product processes; the directory specified by the DOS $TEMP environment variable.
FP_UndoFDKRecording	IntT	Overrides the default value specified in the initialization flag EnableUndoInFDK. Use F_ApiSetInt to set this property value, and F_ApiGetInt to retrieve it. Set the property to zero to disable FDK Undo recording for a session, or to a non-zero value to enable Undo recording.
FP_UserLogin†	StringT	User login name.
FP_UserName†	StringT	User name.

Property	Type	Meaning
FP_Validating	IntT	True if validation is enabled.
FP_VersionMajor†	IntT	Frame version number (before the decimal).
FP_VersionMinor†	IntT	Frame version number (after the decimal).
FP_ViewQuickAccessBar	IntT	True if the QuickAccess bar is visible.
FP_ViewFormattingBar	IntT	True if the formatting bar is visible. FP_ViewFormattingBar is available only on Windows platforms.
FP_WindowSystem†	StringT	Name of window system that the FrameMaker product is running under: MSWindows
FP_FM_XmlDir†	StringT	Directory pathname of $FMHOME/structure/xml.
FP_XSLTTransformationScenarioFile†	StringT	Returns the Transformation File path specified in maker.ini.

Structural elements

The API uses an `FO_Element` object to represent an element and an `FO_ElementDef` object to represent an element definition in a Structured FrameMaker document.

FO_Element

`FO_Element` objects have the following properties.

Element general properties

`FO_Element` objects have the following general properties.

Property	Type	Meaning
`FP_AttrDisplay`	`IntT`	Specifies element's attribute display properties: `FV_ATTR_DISP_NONE`: don't display attributes `FV_ATTR_DISP_REQSPEC`: display required and specified attributes `FV_ATTR_DISP_ALL`: display all attributes
`FP_Attributes`	`F_AttributesT`	The element's attributes.
`FP_ContextLabel`†	`StringT`	The context label (if any) applied to the element.
`FP_ElementIsCollapsed`	`IntT`	`True` if element is collapsed in Structure View.

Property	Type	Meaning
FP_ElementType†	IntT	The type of element: FV_FO_CONTAINER FV_FO_TBL FV_FO_MARKER FV_FO_EQN FV_FO_XREF FV_FO_TBL_TITLE FV_FO_TBL_HEADING FV_FO_TBL_BODY FV_FO_TBL_FOOTING FV_FO_TBL_ROW FV_FO_TBL_CELL FV_FO_FOOTNOTE FV_FO_GRAPHIC FV_FO_SYS_VAR FV_FO_RUBIGROUP FV_FO_RUBI The following values, which were used in previous versions of FrameMaker+SGML, are no longer supported: FV_FO_AFRAME FV_FO_IMP_OBJECT FV_FO_EMPTY FV_FO_EMPTYPGF FV_FO_VAR
FP_MatchingFirstPgf Clauses†	F_IntsT	IDs of the first paragraph clauses (FO_FmtRuleClause IDs) in the element's definition that apply to the element.
FP_FormatOverride†	IntT	True if the element has a format override.
FP_MarkedForNamed Destination	IntT	Used for generatig PDF. If True, this element will have a corresponding Named Destination in the generated PDF.

Property	Type	Meaning
FP_MatchingLastPgf Clauses†	F_IntsT	IDs of the last paragraph clauses (FO_FmtRuleClause IDs) in the element's definition that apply to the element.[a]
FP_MatchingObject Clauses†	F_IntsT	IDs of the object clauses (FO_FmtRuleClause IDs) in the element's definition that apply to the element.
FP_MatchingPrefix Clauses†	F_IntsT	IDs of the prefix clauses (FO_FmtRuleClause IDs) in the element's definition that apply to the element.
FP_MatchingSuffix Clauses†	F_IntsT	IDs of the suffix clauses (FO_FmtRuleClause IDs) in the element's definition that apply to the element.
FP_MatchingText Clauses†	F_IntsT	IDs of the text clauses (FO_FmtRuleClause IDs) in the element's definition that apply to the element.
FP_TextRange†	F_TextRangeT	Text range that the element encompasses (see the explanation below).
FP_Unique†	IntT	The element's UID.
FP_UserString	StringT	A string to which clients can store private data.

a. The FP_Matching*ClauseType*Clauses properties specify only format rule clauses that are in the element definition's format rules (that is, the format rules specified by the element definition's FP_TextFormatRules and FP_ObjectFormatRules properties). Format rule clauses that the element inherits from ancestor elements may also apply to it. To determine whether an element inherits format rule clauses from ancestor elements, you must traverse up the structure tree and check the FP_Matching Clauses properties for each ancestor element.

The `FP_TextRange` property for a structural element specifies an `F_TextRangeT` structure. The text locations specified by the `beg` and `end` fields of the `F_TextRangeT` structure depend on the element type, as shown in the following table.

Element type	Text locations specified by the beg and end fields
`FV_FO_CONTAINER`	`beg` specifies the beginning of the container element, variable, or cross-reference; `end` specifies the end of the container element, variable, or cross-reference
`FV_FO_SYS_VAR`	
`FV_FO_XREF`	
`FV_FO_FOOTNOTE`	`beg` and `end` both specify the anchor of the object (the footnote, marker, table, graphic, or equation)
`FV_FO_MARKER`	
`FV_FO_TBL`	
`FV_FO_GRAPHIC`	
`FV_FO_EQN`	
`FV_FO_TBL_TITLE`	`beg` and `end` both specify nothing (the `F_ApiGetTextRange()` call fails with a `FE_BadOperation` error)
`FV_FO_TBL_HEADING`	
`FV_FO_TBL_BODY`	
`FV_FO_TBL_FOOTING`	
`FV_FO_TBL_ROW`	
`FV_FO_TBL_CELL`	
Text element	`beg` specifies location of the first character (or other content) in the dummy text element; `end` specifies the offset after the last character (or other content) in the element

Element ID properties

FO_Element objects have the following properties that specify the IDs of other objects.

Property	Type	Meaning
FP_BookComponent†	F_ObjHandleT	Component file in book (FO_BookComponent ID).
FP_ElementDef	F_ObjHandleT	Element's element definition (FO_ElementDef ID).
FP_FirstChildElement†	F_ObjHandleT	If element is a container, element's first child element (FO_Element ID).
FP_LastChildElement†	F_ObjHandleT	If element is a container, element's last child element (FO_Element ID).
FP_NextSiblingElement†	F_ObjHandleT	Element's next sibling element (FO_Element ID).
FP_Object†	F_ObjHandleT	ID of the object that an element contains. The type of object the ID specifies depends on the element definition as follows: FV_FO_TBL: FO_Tbl FV_FO_MARKER: FO_Marker FV_FO_EQN: FO_AFrame (containing the equation) FV_FO_XREF: FO_XRef FV_FO_SYS_VAR: FO_Var FV_FO_FOOTNOTE: FO_Fn FV_FO_GRAPHIC: FO_AFrame (containing the graphic) FV_FO_TBL_TITLE: FO_Tbl FV_FO_TBL_HEADING: FO_Tbl FV_FO_TBL_BODY: FO_Tbl FV_FO_TBL_FOOTING: FO_Tbl FV_FO_TBL_ROW: FO_Row FV_FO_TBL_CELL: FO_Cell FV_FO_RUBIGROUP: FO_Rubi FV_FO_RUBI: FO_Rubi

Property	Type	Meaning
FP_ParentElement†	F_ObjHandleT	Element's parent element (FO_Element ID).
FP_PrevSiblingElement†	F_ObjHandleT	Element's previous sibling element (FO_Element ID).

Element validation properties

FO_Element objects have the following validation properties.

Property	Type	Meaning
FP_AllowAsSpecialCase	IntT	True if the element is treated as a special case.
FP_AttributeValue Invalid	IntT	True if the element contains an attribute value that is invalid.
FP_BookComponentMissing†	IntT	True if a component file is missing from a book.
FP_ElementIsUndefined†	IntT	True if the element is undefined.
FP_ErrorInBookComponent†	IntT	True if there is a validation error for a component in a book.
FP_ContentIsLoosely Valid†	IntT	True if the content is loosely valid (it has some missing elements).
FP_ContentIsStrictly Valid†	IntT	True if the content of the element is strictly valid.
FP_ContentMustBeEmpty†	IntT	True if the element can't have any content.
FP_ContentNeededAt Begin†	IntT	True if content is needed at the beginning of the element.
FP_ContentNeededAtEnd†	IntT	True if content is needed at end of the element. FP_ContentNeededAtEnd is obsolete, but is supported for backward compatibility.

Property	Type	Meaning
`FP_ElementIsExcludedIn Context`†	`IntT`	`True` if the element is excluded.
`FP_ElementIsInvalidIn Parent`†	`IntT`	`True` if the element cannot occur anywhere in its current parent.
`FP_ElementIsInvalidIn Position`†	`IntT`	`True` if the element is invalid in its current position.
`FP_HoleBeforeElement`†	`IntT`	`True` if there are one or more missing elements before the element within the same parent.
`FP_InvalidHighestLevel`†	`IntT`	`True` if the element cannot be the highest-level element in the flow.
`FP_NextInvalidElement`†	`F_ObjHandleT`	Next invalid element in the document (`FO_Element` ID).
`FP_TextIsInvalidIn Element`†	`IntT`	`True` if the element contains only text and the element definition disallows it. `FP_TextIsInvalidInElement` is obsolete and is no longer supported.

Property	Type	Meaning
FP_ValidationFlags†	IntT	Bit flags specifying the element's validity. To determine all the ways in which an element is invalid without querying all the validation properties, query this property. Each bit flag in the returned value represents the value of the validation property with the corresponding name. For example, if the FP_ElementTypeMismatch property is True, the FV_ELEM_TYPE_MISMATCH flag is set. FV_ELEM_UNDEFINED FV_ELEM_TYPE_MISMATCH FV_ELEM_EXCLUDED FV_ELEM_INVALID_IN_PARENT FV_ELEM_INVALID_AT_POSITION FV_ELEM_HAS_TEXT_INVALID FV_ELEM_CONTENT_MUST_BE_EMPTY FV_ELEM_MISSING_CONTENT_BEFORE FV_ELEM_MISSING_CONTENT_AT_BEG FV_ELEM_MISSING_CONTENT_AT_END FV_ELEM_NOT_VALID_AS_ROOT FV_ELEM_BOOK_COMP_MISSING FV_ELEM_BOOK_COMP_INVALID FV_ELEM_ATTRVAL_REQUIRED FV_ELEM_ATTRVAL_INVALID FV_ELEM_CONTENT_STRICTLY_VALID FV_ELEM_CONTENT_LOOSELY_VALID

Element DITA properties

`FO_Element` objects have the following DITA-related properties.

Property	Type	Meaning
`FP_NextDITAConrefElementInDoc`	`StringT`	The next Conref element in the document
`FP_NextDITAXrefElementInDoc`	`StringT`	The next XRef element in the document
`FP_NextDITALinkElementInDoc`	`StringT`	The next Link element in the document
`FP_NextDITATopicrefElementInDoc`	`StringT`	The next Topicref element in the document
`FP_NextDITATopicsetrefElementInDoc`	`StringT`	The next Topicsetref element in the document

FO_ElementDef

`FO_ElementDef` objects have the following properties.

Property	Type	Meaning
`FP_AlsoInsert`	`F_StringsT`	The list of the tags of child elements that are automatically inserted when an element is initially added.. For example, consider the case where you set `AlsoInserts` property of element definition of A to [[a1, a11],[a2, a21]]. In this case, when A is inserted, the following elements are inserted: children of A are a1 and a2, and children of a1 and a2 are a11 and a21 respectively.
`FP_AttributeDefs`	`F_AttributeDefsT`	The element definition's attribute definitions.
`FP_BannerText`	`StringT`	Text string of banner text associated with an element definition object.
`FP_Comment`	`StringT`	Text string of comment.

Property	Type	Meaning
FP_DescriptiveTag	StringT	A small description about the element. If specified in the EDD, the user ca n view element description in Element Catalog of the structured document.
FP_InitStructure Pattern	StringT	The initial structure pattern; for table elements, a comma delimited string that specifies the necessary child elements to automatically insert.

Property	Type	Meaning
FP_ElementDefType	IntT	Type of formatter object represented by the element with element definition. FV_FO_CONTAINER identifies a container element. Other values identify object (noncontainer) elements. FV_FO_UNSPECIFIED FV_FO_CONTAINER FV_FO_TBL FV_FO_MARKER FV_FO_EQN FV_FO_XREF FV_FO_TBL_TITLE FV_FO_TBL_HEADING FV_FO_TBL_BODY FV_FO_TBL_FOOTING FV_FO_TBL_ROW FV_FO_TBL_CELL FV_FO_FOOTNOTE FV_FO_GRAPHIC FV_FO_SYS_VAR The following values, which were used in previous versions of FrameMaker+SGML, are no longer supported: FV_FO_AFRAME FV_FO_IMP_OBJECT FV_FO_EMPTY FV_FO_EMPTYPGF FV_FO_VAR FV_FO_RUBIGROUP FV_FO_RUBI
FP_ElementIn Catalog	IntT	True if the element is in the Element Catalog.
FP_ElementPgf Format	StringT	The name of the paragraph format applied to the element.
FP_Exclusions	F_StringsT	List of excluded elements.

Property	Type	Meaning
FP_FirstPgfRules†	F_IntsT	The IDs of the first paragraph format rules (FO_FmtRule IDs).[a]
FP_GeneralRule	StringT	Text of the element's general rule.
FP_GeneralRule ErrorOffsets†	F_IntsT	Contains the error offsets (two positions are specified only if the content rule is ambiguous).
FP_Inclusions	F_StringsT	List of included elements.
FP_LastPgfRules†	F_IntsT	The IDs of the last paragraph format rules (FO_FmtRule IDs).
FP_Name†	StringT	Name of the element definition.
FP_NextElementDefIn Doc†	F_ObjHandleT	Next element definition in the document's list of element definitions (FO_ElementDef ID).
FP_ObjectFmtRules†	F_IntsT	The IDs of the object format rules (FO_FmtRule IDs).
FP_ObjectType	IntT	FP_ObjectType is obsolete and no longer supported. Use FP_ElementDefType instead.
FP_PrefixRules†	F_IntsT	The IDs of the prefix format rules (FO_FmtRule IDs).
FP_SuffixRules†	F_IntsT	The IDs of the suffix format rules (FO_FmtRule IDs).
FP_TableTagging	StringT	If the element is a table, the table format for new instances of the element.
FP_TextFmtRules†	F_IntsT	The IDs of the text format rules (FO_FmtRule IDs).
FP_ValidHighest Level	IntT	True if the element can be used as the highest-level element for a flow

a. To set the format rules for an element definition, use F_ApiNewFmtRuleObject().

Tables

The API uses FO_Cell, FO_Row, and FO_Tbl objects to represent tables.

FO_Cell

The API uses an `FO_Cell` object to represent each cell in a table. `FO_Cell` objects have the following properties.

Property	Type	Meaning
`FP_CellAboveInCol`†	`F_ObjHandleT`	Cell above current cell (`FO_Cell` ID).
`FP_CellAngle`	`IntT`	Angle of rotation.
`FP_CellBelowInCol`†	`F_ObjHandleT`	Cell below current cell (`FO_Cell` ID).
`FP_CellColNum`†	`IntT`	Cell's column number.
`FP_CellDefaultBottom Ruling`†	`F_ObjHandleT`	Cell's default bottom ruling (`FO_RulingFmt` ID).
`FP_CellDefaultLeft Ruling`†	`F_ObjHandleT`	Cell's default left ruling (`FO_RulingFmt` ID).
`FP_CellDefaultRight Ruling`†	`F_ObjHandleT`	Cell's default right ruling (`FO_RulingFmt` ID).
`FP_CellDefaultTop Ruling`†	`F_ObjHandleT`	Cell's default top ruling (`FO_RulingFmt` ID).
`FP_CellIsShown`†	`IntT`	`True` if the cell is conditional and is visible.
`FP_CellIsStraddled`†	`IntT`	If the cell is in a straddle but is not the first cell, `FP_CellIsStraddled` specifies `True`. If the cell is the first cell in a straddle, or it is not in a straddle, `FP_CellIsStraddled` specifies `False`.
`FP_CellNumColsStraddled`†	`IntT`	If the cell is the first cell in a horizontal straddle, `FP_CellNumColsStraddled` specifies the number of columns in the straddle. Otherwise, it specifies `1`.
`FP_CellNumRowsStraddled`†	`IntT`	If the cell is the first cell in a vertical straddle, `FP_CellNumRowsStraddled` specifies the number of rows in the straddle. Otherwise, it specifies `1`.

Property	Type	Meaning
FP_ContentHeight†	MetricT	The distance between the top of the cell and the baseline of the last line in the cell.
FP_CellOverrideBottom Ruling	F_ObjHandleT	Cell's bottom ruling (FO_RulingFmt ID). NULL if there is no override.
FP_CellOverrideFill	IntT	Fill pattern. NULL if there is no override fill pattern.
FP_CellOverrideLeft Ruling	F_ObjHandleT	Cell's left ruling (FO_RulingFmt ID). NULL if there is no override left ruling.
FP_CellOverrideRight Ruling	F_ObjHandleT	Cell's right ruling (FO_RulingFmt ID). NULL if there is no override right ruling.
FP_CellOverride Shading	F_ObjHandleT	Spot color (FO_Color ID). NULL if there is no override shading.
FP_CellOverrideTop Ruling	F_ObjHandleT	Cell's top ruling (FO_RulingFmt ID). NULL if there is no override top ruling.
FP_CellRow†	F_ObjHandleT	Row containing the cell (FO_Row ID).
FP_CellUseOverrideB Ruling	IntT	True if the cell's bottom ruling (specified by FP_CellOverrideBottom Ruling) overrides the default ruling specified by the table format.
FP_CellUseOverrideFill	IntT	True if the cell's fill pattern (specified by FP_CellOverrideFill) overrides the default fill pattern specified by the table format.
FP_CellUseOverride LRuling	IntT	True if the cell's left ruling (specified by FP_CellOverrideLeft Ruling) overrides the ruling specified by the table format.

Property	Type	Meaning
FP_CellUseOverrideRRuling	IntT	True if the cell's right ruling (specified by FP_CellOverrideRightRuling) overrides the ruling specified by the table format.
FP_CellUseOverrideShading	IntT	True if the cell's shading (specified by FP_CellOverrideShading) overrides the default shading specified by the table format.
FP_CellUseOverrideTRuling	IntT	True if the cell's top ruling (specified by FP_CellOverrideTopRuling) overrides the default top ruling specified by the table format.
FP_Element†	F_ObjHandleT	If the cell is in a Structured FrameMaker document, the ID of the element containing the cell.
FP_FirstPgf†	F_ObjHandleT	First paragraph in the cell (FO_Pgf ID).
FP_HighestLevelElement†	F_ObjHandleT	Cell's highest-level element if the cell is in a structured document (FO_Element ID). FP_HighestLevelElement is obsolete but is supported for backward compatibility.
FP_InTextFrame†	F_ObjHandleT	Text frame containing the cell (FO_TextFrame ID).
FP_InTextObj†	F_ObjHandleT	Text object containing the cell (FO_SubCol ID).
FP_LastPgf†	F_ObjHandleT	Last paragraph in the cell (FO_Pgf ID).
FP_NextCellInRow†	F_ObjHandleT	Next cell in current row from left to right (FO_Cell ID).
FP_NextCellInTbl†	F_ObjHandleT	Next cell from left to right (FO_Cell ID). If the cell is at the end of a row, the next cell is the first cell in the next row.
FP_NextCell	F_ObjHandleT	Next cell in the text frame (FO_Cell ID).

Property	Type	Meaning
FP_Overflowed†	IntT	Specifies whether the text in the cell overflows. True if the row Height Limit Maximum is too low to display all the text in the cell.
FP_PrevCellInRow†	F_ObjHandleT	Previous cell in current row (FO_Cell ID).
FP_PrevCell†	F_ObjHandleT	Previous cell in the text frame (FO_Cell ID).
FP_Unique†	IntT	The cell's UID
Text		To get the text in a table cell, call F_ApiGetText() with objId set to the cell ID. To add text to a cell, call F_ApiAddText(). To delete text from a cell, call F_ApiDeleteText(). For more information, see "F_ApiAddText()" on page 80, "F_ApiDeleteText()" on page 151, and "F_ApiGetText()" on page 249.

FO_Row

The API uses an FO_Row object to represent each row in a table. FO_Row objects have the following properties.

Property	Type	Meaning
FP_CondFmtIsShown	IntT	True if the condition is shown.
FP_Element†	F_ObjHandleT	If the row is in a Structured FrameMaker document, the ID of the element containing the row.
FP_FirstCellInRow†	F_ObjHandleT	First cell in row (FO_Cell ID).
FP_Height†	MetricT	Height of the row.
FP_InCond	F_IntsT	Condition tags for row (array of FO_CondFmt IDs).
FP_LocX†	MetricT	Offset from the left side of the text frame containing the row.

Property	Type	Meaning
FP_LocY†	MetricT	Offset from the top of the page frame containing the row.
FP_NextRowInTbl†	F_ObjHandleT	Next row (FO_Row ID) in the table.
FP_PrevRowInTbl†	F_ObjHandleT	Previous row (FO_Row ID) in the table.
FP_RowIsShown†	IntT	True if the conditional row is shown.
FP_RowKeepWithNext	IntT	True if Keep With Next Row is enabled.
FP_RowKeepWithPrev	IntT	True if Keep With Previous Row is enabled.
FP_RowMaxHeight	MetricT	Maximum row height.
FP_RowMinHeight	MetricT	Minimum row height.
FP_RowStart	IntT	Row placement: FV_ROW_ANYWHERE FV_ROW_TOP_OF_COL FV_ROW_TOP_OF_PAGE FV_ROW_TOP_OF_LEFT_PAGE FV_ROW_TOP_OF_RIGHT_PAGE
FP_RowTbl†	F_ObjHandleT	Table containing the row (FO_Tbl ID).
FP_RowType†	IntT	Type of row: FV_ROW_HEADING FV_ROW_BODY FV_ROW_FOOTING
FP_SepOverride	F_ObjHandleT	Color separation format override (FO_Color ID).
FP_StyleOverrides	IntT	Style condition indicators for conditional text: FV_CS_DOUBLE_UNDERLINE FV_CS_NO_OVERRIDE FV_CS_OVERLINE FV_CS_SINGLE_UNDERLINE FV_CS_STRIKETHROUGH All style condition indicators are represented as hatched lines for the table rows.

Property	Type	Meaning
FP_UseSepOverride	IntT	True if FP_SepOverride property overrides default from the table.
FP_Width†	MetricT	Width of the row.

FO_Tbl

FO_Tbl objects have the following properties.

Table basic properties

FO_Tbl objects have the following properties that specify a table's indents, alignment, and other placement characteristics.

Property	Type	Meaning
FP_ContentHeight†	MetricT	The height of the table title.
FP_Locked	IntT	True if the table is part of a text inset that retains formatting information from the source document. The table is not affected by global formatting performed on the document.
FP_OrphanRows	IntT	Number of orphan rows.
FP_Overflowed†	IntT	True if the table has cells that are not shown because they extend beyond the text frame boundaries.
FP_TblAlignment	IntT	Horizontal placement of table: FV_ALIGN_TBL_CENTER FV_ALIGN_TBL_LEFT FV_ALIGN_TBL_RIGHT
FP_TblCellBottom Margin	MetricT	Default bottom cell margin for the table
FP_TblCellLeft Margin	MetricT	Default left cell margin for the table
FP_TblCellRight Margin	MetricT	Default right cell margin for the table
FP_TblCellTop Margin	MetricT	Default top cell margin for the table

Property	Type	Meaning
FP_TblLeftIndent	MetricT	Left indent for the table
FP_TblPlacement	IntT	Vertical placement of table on page: FV_TBL_ANYWHERE FV_TBL_TOP_OF_COL FV_TBL_TOP_OF_PAGE FV_TBL_TOP_OF_LEFT_PAGE FV_TBL_TOP_OF_RIGHT_PAGE FV_TBL_FLOAT
FP_TblRightIndent	MetricT	Right indent for the table
FP_TblSpaceAbove	MetricT	Vertical space above the table
FP_TblSpaceBelow	MetricT	Vertical space below the table
FP_TextLoc	F_TextLocT	The text location of the table's anchor
FP_Unique†	IntT	The table's UID

Table general properties

FO_Tbl objects have the following properties that provide information about a table's width and its columns and rows.

Property	Type	Meaning
FP_FirstRowInTbl†	F_ObjHandleT	First row in the table (FO_Row ID)
FP_LastRowInTbl†	F_ObjHandleT	Last row in the table (FO_Row ID)
FP_NextTblInDoc†	F_ObjHandleT	Next table (FO_Tbl ID) in the document
FP_TblCatalogEntry†	IntT	True if the table's format is in the Table Catalog
FP_TblColWidths	F_MetricsT	List of column widths
FP_TblNumbering	IntT	Direction of autonumbering for the table: FV_TBL_NUM_BY_COL FV_TBL_NUM_BY_ROW
FP_TblNumCols†	IntT	Number of columns in the table

Property	Type	Meaning
FP_TblNumRows†	IntT	Number of rows in the table
FP_TblTag	StringT	Name of table format
FP_TblWidth†	MetricT	Horizontal width of the table

Table ruling properties

FO_Tbl objects have the following properties that specify the table rulings.

Property	Type	Meaning
FP_TblBodyRowRuling	F_ObjHandleT	Ruling applied to body rows specified by FP_TblBodyRowRulingPeriod (FO_RulingFmt ID).
FP_TblBodyRowRulingPeriod	IntT	The periodicity of the ruling specified by FP_TblBodyRowRuling. For example, if FP_TblBodyRowRulingPeriod is set to 3, the ruling specified by FP_TblBodyRowRuling is applied to every third row.
FP_TblBottomRuling	IntT	Ruling applied to the bottom of the table (FO_RulingFmt ID).
FP_TblColRuling	F_ObjHandleT	Ruling applied to table columns specified by FP_TblColRulingPeriod (FO_RulingFmt ID).
FP_TblColRulingPeriod	IntT	The periodicity of the ruling specified by FP_TblColRuling. For example, if FP_TblColRulingPeriod is set to 2, the ruling specified by FP_TblColRuling is applied to every other column.
FP_TblHFRowRuling	F_ObjHandleT	Ruling for table heading and footing rows (FO_RulingFmt ID).
FP_TblHFSeparatorRuling	F_ObjHandleT	Separator ruling for table heading and footing rows (FO_RulingFmt ID).

Property	Type	Meaning
FP_TblLastBodyRuling	IntT	True if Draw Bottom Ruling on Last Sheet Only is enabled (FO_RulingFmt ID).
FP_TblLeftRuling	F_ObjHandleT	Ruling for the left side of the table (FO_RulingFmt ID).
FP_TblOtherBodyRow Ruling	F_ObjHandleT	Ruling for body rows that aren't specified by FP_TblBodyRowRulingPeriod (FO_RulingFmt ID).
FP_TblOtherColRuling	F_ObjHandleT	Ruling for columns that aren't specified by FP_TblColRulingPeriod (FO_RulingFmt ID).
FP_TblRightRuling	F_ObjHandleT	Ruling for the right side of the table (FO_RulingFmt ID).
FP_TblTopRuling	F_ObjHandleT	Ruling for the top of the table (FO_RulingFmt ID).

Table selection properties

FO_Tbl objects have the following properties that specify a table's selected rows and columns. All table selection properties are read-only. To select table rows or cells, use F_ApiMakeTblSelection(). For more information, see "F_ApiMakeTblSelection()" on page 297.

Property	Type	Meaning
FP_BottomRow Selection†	F_ObjHandleT	Bottom body row in selection, if table is selected (FO_Row ID)
FP_LeftColNum†	IntT	Number of leftmost selected column, if table is selected (columns are numbered from left to right, starting with 0)
FP_RightColNum†	IntT	Number of rightmost selected column, if table is selected (columns are numbered from left to right, starting with 0)

Property	Type	Meaning
FP_TblTitle Selected†	IntT	True if table title is selected
FP_TopRow Selection†	F_ObjHandleT	Top row in selection, if table is selected (FO_Row ID)

Table shading and color properties

FO_Tbl objects have the following properties that specify a table's shading.

Property	Type	Meaning
FP_TblBodyFirstColor	F_ObjHandleT	First spot color for table body (FO_Color ID)
FP_TblBodyFirstFill	IntT	First fill pattern for table body
FP_TblBodyFirstPeriod	IntT	Number of columns or body rows to which the first fill pattern (specified by FP_TblBodyFirstFill) is applied
FP_TblBodyNextColor	F_ObjHandleT	Exception color for columns or body rows (FO_Color ID)
FP_TblBodyNextFill	IntT	Exception fill pattern for table body
FP_TblBodyNextPeriod	IntT	Number of columns or body rows to which the exception fill pattern (specified by FP_TblBodyNextFill) is applied
FP_TblBodyShadeBy	IntT	True if Shade By is set to Columns; False if Shade By is set to Rows
FP_TblHFColor	F_ObjHandleT	Color for table heading and footing
FP_TblHFFill	IntT	Fill pattern for table heading and footing (integer percentage)

Table structure properties

FO_Tbl objects have the following properties in structured documents.

Property	Type	Meaning
FP_Element†	F_ObjHandleT	The ID of the element associated with the table
FP_TblBodyElement†	F_ObjHandleT	The ID of the element containing the table's body rows
FP_TblElement†	F_ObjHandleT	The ID of the element containing the table
FP_TblFooterElement†	F_ObjHandleT	The ID of the element containing the table's footer rows
FP_TblHeaderElement†	F_ObjHandleT	The ID of the element containing the table's header rows
FP_TblTitleElement†	F_ObjHandleT	The ID of the element containing the table title

Table title properties

FO_Tbl objects have the following properties that specify a table title's characteristics.

Property	Type	Meaning
FP_FirstPgf†	F_ObjHandleT	If table has a title, the first paragraph in the title (FO_Pgf ID).
FP_HighestLevelElement†	F_ObjHandleT	If table is in a structured document and has a title, the title's highest-level element (FO_Element ID). FP_HighestLevelElement is obsolete but is supported for backward compatibility.
FP_LastPgf†	F_ObjHandleT	If table has a title, the last paragraph in the title (FO_Pgf ID).
FP_TblTitleGap	MetricT	Gap between the title and top or bottom row of the table.

Property	Type	Meaning
`FP_TblTitle Position`	`IntT`	The placement of the table title: `FV_TBL_NO_TITLE`: table has no title `FV_TBL_TITLE_BELOW`: the title appears below the table `FV_TBL_TITLE_ABOVE`: the title appears above the table
`FP_TblTitle Selected`†	`IntT`	`True` if the table title is selected.

Table formats

The API uses an `FO_TblFmt` object to represent each table format in a document.

FO_TblFmt

`FO_TblFmt` objects have the following properties.

Table format basic properties

`FO_TblFmt` objects have the following properties that specify a table's indents, alignment, and other placement characteristics. Note that a FO_TblFmt object also includes all the properties for a FO_PgfFmt object in order to describe the properties of the table title. See "FO_PgfFmt" on page 887

Property	Type	Meaning
`FP_OrphanRows`	`IntT`	Number of orphan rows

`FP_TblAlignment`	`IntT`	Horizontal placement of table:
		`FV_ALIGN_TBL_CENTER`
		`FV_ALIGN_TBL_LEFT`
		`FV_ALIGN_TBL_RIGHT`

`FP_TblCellBottom Margin`	`MetricT`	Default bottom cell margin for the table.
`FP_TblCellLeft Margin`	`MetricT`	Default left cell margin for the table.
`FP_TblCellRight Margin`	`MetricT`	Default right cell margin for the table.
`FP_TblCellTop Margin`	`MetricT`	Default top cell margin for the table.
`FP_TblLeftIndent`	`MetricT`	Left indent for the table
`FP_TblLeftIndent`	`MetricT`	Left indent for table

Property	Type	Meaning
FP_TblPlacement	IntT	Vertical placement of table on page: FV_TBL_ANYWHERE FV_TBL_TOP_OF_COL FV_TBL_TOP_OF_PAGE FV_TBL_TOP_OF_LEFT_PAGE FV_TBL_TOP_OF_RIGHT_PAGE FV_TBL_FLOAT
FP_TblRightIndent	MetricT	Right indent for table
FP_TblSpaceAbove	MetricT	Vertical space above table
FP_TblSpaceBelow	MetricT	Vertical space below table
FP_TblTitleGap	MetricT	Gap between title and top or bottom row
FP_TblTitle Position	IntT	Placement of table title: FV_TBL_NO_TITLE FV_TBL_TITLE_BELOW FV_TBL_TITLE_ABOVE

Table format general properties

FO_TblFmt objects have the following properties.

Property	Type	Meaning
FP_TblCatalogEntry	IntT	True if format is in the Table Catalog
FP_Name	StringT	Name of the paragraph format of the table title
FP_NextTblFmtInDoc†	F_ObjHandleT	Next table format in the document (FO_TblFmt ID)
FP_TblNumbering	IntT	Direction of autonumbering for the table: FV_TBL_NUM_BY_COL FV_TBL_NUM_BY_ROW
FP_TblTag	StringT	Name of the table format

Table format new table properties

FO_TblFmt objects have the following properties that specify how many rows and columns a new table will have when it is created.

Property	Type	Meaning
FP_TblInitNumBodyRows	IntT	Number of body rows for new table
FP_TblInitNumCols	IntT	Number of columns for new table
FP_TblInitNumFRows	IntT	Number of footing rows for new table
FP_TblInitNumHRows	IntT	Number of heading rows for new table

Table format ruling properties

FO_TblFmt objects have the following properties that specify a table's rulings.

Property	Type	Meaning
FP_TblBodyRowRuling	F_ObjHandleT	Ruling for body rows that aren't specified by FP_TblBodyRowRulingPeriod (FO_RulingFmt ID).
FP_TblBodyRowRulingPeriod	IntT	The periodicity of the ruling specified by FP_TblOtherBodyRowRuling. For example, if FP_TblBodyRowRulingPeriod is set to 3, the ruling specified by FP_TblOtherBodyRowRuling is applied to every third row.
FP_TblBottomRuling	F_ObjHandleT	Ruling for the bottom of the table (FO_RulingFmt ID).
FP_TblColRuling	F_ObjHandleT	Ruling for columns that aren't specified by FP_TblColRulingPeriod (FO_RulingFmt ID).
FP_TblColRulingPeriod	IntT	The periodicity of the ruling specified by FP_TblOtherColRuling. For example, if FP_TblColRulingPeriod is set to 2, the ruling specified by FP_TblOtherColRuling is applied to every other column.

Property	Type	Meaning
FP_TblHFRowRuling	F_ObjHandleT	Ruling for the heading and footing rows (FO_RulingFmt ID).
FP_TblHFSeparator Ruling	F_ObjHandleT	Separator ruling for the table heading and footing rows (FO_RulingFmt ID).
FP_TblLastBodyRuling	IntT	True if Draw Bottom Ruling on Last Sheet Only is enabled (FO_RulingFmt ID).
FP_TblLeftRuling	F_ObjHandleT	Ruling for the left side of the table (FO_RulingFmt ID).
FP_TblOtherBodyRow Ruling	F_ObjHandleT	Ruling applied to body rows specified by FP_TblBodyRowRulingPeriod (FO_RulingFmt ID).
FP_TblOtherColRuling	F_ObjHandleT	Ruling applied to table columns specified by FP_TblColRulingPeriod (FO_RulingFmt ID).
FP_TblRightRuling	F_ObjHandleT	Ruling for the right side of the table (FO_RulingFmt ID).
FP_TblTopRuling	F_ObjHandleT	Ruling for the top of the table (FO_RulingFmt ID).

Table format shading and color properties

FO_TblFmt objects have the following properties that specify a table's shading.

Property	Type	Meaning
FP_TblBodyFirstColor	F_ObjHandleT	First spot color for table body (FO_Color ID)
FP_TblBodyFirstFill	IntT	First fill pattern for table body
FP_TblBodyFirstPeriod	IntT	Number of columns or body rows to which the first fill pattern (specified by FP_TblBodyFirstFill) is applied
FP_TblBodyNextColor	F_ObjHandleT	Exception color for columns or body rows (FO_Color ID)

Property	Type	Meaning
FP_TblBodyNextFill	IntT	Exception fill pattern for table body
FP_TblBodyNextPeriod	IntT	Number of columns or body rows to which the exception fill pattern (specified by FP_TblBodyNextFill) is applied
FP_TblBodyShadeBy	IntT	True if Shade By is set to Columns; False if Shade By is set to Rows
FP_TblHFColor	F_ObjHandleT	Color for table heading and footing
FP_TblHFFill	IntT	Fill pattern for table heading and footing (integer percentage)

Text columns

The API uses FO_TextFrame objects to represent text frames and FO_SubCol objects to represent the text columns in them. For more information, see "Columns" on page 757 and "FO_TextFrame" on page 863.

Text frames

Text frames are a type of graphics. For information on text frames, see "FO_TextFrame" on page 863.

Text insets

The API uses FO_TiApiClient, FO_TiFlow, FO_TiText, and FO_TiTextTable objects to represent text that is imported by reference (text insets).

Each type of text inset object has a set of properties that are common to all text inset objects and a set of properties that are specific to it.

Common text inset properties

The following properties are common to all text inset objects.

Property	Type	Meaning
FP_ImportHint	StringT	Record identifying the filter used to import the text. The FrameMaker product uses this record to find the filter to use when updating the inset. For a complete description of the syntax of this string, see "Syntax of FP_ImportHint strings" on page 939.
FP_TiLocked	IntT	True if the inset is locked. To change an inset's contents, you must unlock it. Always relock an inset after you have finished changing its contents.
FP_Name	StringT	A name assigned to the inset by an FDK client. It is not automatically assigned by the FrameMaker product.
FP_NextTiInDoc†	F_ObjHandleT	The ID of the next text inset in the list of text insets in the document (FO_TiApiClient, FO_TiText, FO_TiTextTable, or FO_TiFlow ID).
FP_TextRange†	F_TextRangeT	The text range, in the document containing the text inset, occupied by the text inset.
FP_TiAutomaticUpdate	IntT	True if the inset is updated automatically. FP_TiAutomaticUpdate has no effect if the document's FP_DontUpdateTextInsets property is set to True.
FP_TiFile	StringT	Pathname of the source file.
FP_TiFileModDate	StringT	The modification date of the text inste's source file.
FP_TiLastUpdate	IntT	Time when the inset was last updated, expressed in seconds since 1 January, 1970.
†		
FP_Unique†	IntT	The text inset's UID.

Syntax of FP_ImportHint strings

The `FP_ImportHint` property specifies a record that identifies the filter used to import text by reference. FrameMaker products use this record to find the correct filter to update a text inset. The syntax of this record is:

`record_vers vendor format_id platform filter_vers filter_name`

Note that the fields in the record are not separated by spaces.

```
0001XTNDWDBNMACP0002MS Word 4,5
```

IMPORTANT: *If you are setting an* `FP_ImportHint` *property, the string you are using should be terminated with* `NULL`*. The string itself must not contain* `NULL` *or undisplayable characters.*

You can determine which `FP_ImportHint` strings are registered by querying the `FP_ExportFilters` property of the `FO_Session` object.

Each field of the record (except *`filter_name`*) specifies a four-byte code. If a code contains fewer than four alphanumeric characters, the remaining bytes must be filled out with spaces.

The rest of this section describes each field in the record.

`record_vers` specifies the version of the record, currently 0001.

`vendor` is a code specifying the filter's vendor. The code is a string of four characters. The following table lists some possible codes.

Code	Meaning
`PGRF`	Built-in Frame filters
`FAPI`	External Frame FDK client filter
`FFLT`	External Frame filters
`IMAG`	External ImageMark filters
`XTND`	External XTND filters

Note that this is not a comprehensive list of codes. Codes may be added to this list by Frame or by developers at your site.

format_id is a code specifying the format that the filter translates. The code is a string of four characters. The following table lists the possible codes.

Code	Meaning
HTML	HTML document (for export, only)
XML	XML document (for export, only)
WDBN	Microsoft Word compound document
WPBN	WordPerfect compound document
RTF	Microsoft's RTF compound document
IAF	Interleaf compound document
MRTF	MIF to RTF export
MIAF	MIF to IAF export
MWPB	MIF to WordPerfect export
MML	Maker Markup Language
CVBN	Corel Ventura compound document
	Corel Ventura compound document (Windows)
TEXT	Plain text
TXIS	Text ISO Latin 1
TANS	Text ANSI (Windows)
TMAC	Text (Macintosh)
TASC	Text ASCII
TJIS	Japanese JIS
TSJS	Japanese Shift-JIS

Code	Meaning
TEUJ	Japanese EUC
TBG5	Traditional Chinese BIG 5
TEUH	Traditional Chinese EUC-CNS
TXHZ	Simplified Chinese HZ
TXGB	Simplified Chinese GB
TKOR	Korean

Note that this is not a comprehensive list of codes. Codes may be added to this list by Frame or by developers at your site.

platform is a code specifying the platform on which the filter was run. The code is a string of four characters. The following table lists the possible codes.

Code	Meaning
WINT	Windows NT
WIN3	Windows 3.1
WIN4	Windows 95

ilter_vers is a string of four characters identifying the version of the filter on that platform. For example, version 1.0 of a filter is represented by the string `1.0` or the string `0001`.

filter_name is a C string (up to 31 characters long) that describes the filter.

For example, the following two records specify the HTML and XML filters on the Windows 95 platform:

```
0001ADBEHTMLWIN40001HTML
0001ADBEXMLWIN40001XML
```

FO_TiApiClient properties

An `FO_TiApiClient` object represents text imported by an FDK client. `FO_TiApiClient` objects have the following properties.

Property	Type	Meaning
Common text inset properties		See page 938.
`FP_TiClientData`	`StringT`	Data used by the client (for example, an SQL query).
`FP_TiClientName`	`StringT`	The registered name of the client that created the inset.
`FP_TiClientSource`	`StringT`	The name that appears as the source in the Text Inset Properties dialog box.
`FP_TiClientType`	`StringT`	The name that appears as the source type in the Text Inset Properties dialog box.
`FP_TiIsUnresolved`	`IntT`	`True` if the inset is unresolved. A client should set this property to `True` if it is unable to resolve the inset.

FO_TiFlow properties

An `FO_TiFlow` object represents text imported from a FrameMaker product document or a MIF file. `FO_TiFlow` objects have the following properties.

Property	Type	Meaning
Common text inset properties		See page 938.
`FP_TiFlowName`	`StringT`	The name of the imported flow if `FP_TiMainFlow` is `False`.

`FP_TiFlowPageSpace`	`IntT`	The type of pages the imported flow is on:
		`FV_BODY_PAGE`
		`FV_REFERENCE_PAGE`

Property	Type	Meaning
FP_TiFormat	IntT	Source of the imported text's format: FV_SourceDoc: the text is formatted with formats from the source document FV_PlainText: the text is formatted as plain text FV_EnclosingDoc: the text is formatted with formats from the document into which it is imported
FP_TiMainFlow	IntT	True if the inset text is imported from the main flow of the source document.
FP_TiRemovePageBreaks	IntT	True if page breaks are removed from the text when FP_TiFormat is set to FV_EnclosingDoc.
FP_TiRemoveOverrides	IntT	True if format overrides are removed from the text when FP_TiFormat is set to FV_EnclosingDoc.

FO_TiText properties

An FO_TiText object represents text imported from a text file. FO_TiText objects have the following properties.

Property	Type	Meaning
Common text inset properties		See page 938.
FP_TiEOLisEOP	IntT	True if line ends in the imported text file are treated as paragraph ends.
FP_TiTextEncoding†	StringT	The FP_ImportHintString for the text inset. If this is not a FO_TiText or FO_TiTextTable, the string is null.

FO_TiTextTable properties

An `FO_TiTextTable` object represents text imported from a text file into a table. `FO_TiTextTable` objects have the following properties.

Property	Type	Meaning
Common text inset properties		See page 938.
`FP_TiByRows`	`IntT`	`True` if each paragraph in the imported text is converted to a row of table cells; `False` if each paragraph in the imported text is converted to a table cell.
`FP_TiTblTag`	`StringT`	The table format tag of the imported table.
`FP_TiHeadersEmpty`	`IntT`	`True` if the imported text is not used to fill the heading rows.
`FP_TiNumSeparators`	`IntT`	If `FP_TiSeparator` specifies a space, the number of spaces used as a separator to parse the text into table cells.
`FP_TiSeparator`	`StringT`	If `FP_TiByRows` is `True`, a string specifying a separator, such as a tab, used to parse the text into table cells.
`FP_TiNumCols`	`IntT`	If `FP_TiByRows` is `False`, the number of columns in the table.
`FP_TiNumHeaderRows`	`IntT`	The number of heading rows in the table.
`FP_TiTextEncoding`†	`StringT`	The `FP_ImportHintString` for the text inset. If this is not a `FO_TiText` or `FO_TiTextTable`, the string is null.

Text properties

Text has the following properties. To retrieve these properties for a text location, use `F_ApiGetTextPropVal()` or `F_ApiGetTextProps()`. The properties these functions return apply to the character to the right of the location you specify.

To set text properties for a text range, use `F_ApiSetTextPropVal()` or `F_ApiSetTextProps()`.

Property	Type	Meaning

`FP_Capitalization`	`IntT`	Type of capitalization:
		`FV_CAPITAL_CASE_NORM`
		`FV_CAPITAL_CASE_SMALL`
		`FV_CAPITAL_CASE_LOWER`
		`FV_CAPITAL_CASE_UPPER`

`FP_ChangeBar`	`IntT`	`True` if Change Bars is enabled at the text location.
`FP_CharTag`	`StringT`	Name of character format tag applied to text location.
`FP_Color`	`F_ObjHandleT`	Spot color (`FO_Color` ID).
`FP_CombinedFontFamily`	`F_ObjHandleT`	Combined font definition (`FO_CombinedFontDefn`)
`FP_FontAngle`	`IntT`	Font angle (specifies an index into the array of font angles provided by the session property `FP_FontAngleNames`).
`FP_FontFamily`	`IntT`	Font family (specifies an index into the array of font families provided by the session property `FP_FontFamilyNames`).
`FP_Font PlatformName`	`StringT`	Name that uniquely identifies a font on a specific platform (for more information, see “How the API represents fonts” on page 107 of the *FDK Programmer’s Guide*).
`FP_FontPost ScriptName`	`StringT`	Name given to a font when it is sent to a PostScript printer (for more information, see “How the API represents fonts” on page 107 of the *FDK Programmer’s Guide*).
`FP_FontSize`	`MetricT`	Font size (2 pt to 400 pt).
`FP_FontVariation`	`IntT`	Font variation (specifies an index into the array of font variations provided by the session property `FP_FontVariationNames`).

Property	Type	Meaning
FP_FontWeight	IntT	Font weight (specifies an index into the array of font weights provided by the session property FP_FontWeightNames).
FP_Height†	MetricT	Height of text at text location.
FP_InCond	F_IntsT	Condition tags that apply to the text (array of FO_CondFmt IDs).
FP_InTextFrame†	F_ObjHandleT	Text frame containing the text (FO_TextFrame ID).
FP_InTextObj†	F_ObjHandleT	Text frame or text line in which text appears (FO_TextFrame or FO_TextLine ID).
FP_KernX	MetricT	Horizontal kern value for manual kerning expressed as a percentage of an em (metric –100% to 1000%).[a] A positive value moves a character right, and a negative value moves a character left.
FP_KernY	MetricT	Vertical kern value for manual kerning expressed as a percentage of an em (metric –100% to 1000%). A positive value moves characters up, and a negative value moves characters down.
FP_LineAscent†	MetricT	The ascent of the line containing the text, measured from the line's baseline.
FP_LineBaseLine†	MetricT	The location of the line containing the text, measured from the top of the object containing the text.
FP_LineDescent†	MetricT	The descent of the line containing the text, measured from the line's baseline.
FP_Locked	IntT	True if the text is included in a text inset that gets its formatting from the source document.
FP_LocX†	MetricT	Offset of the left side of a character from the left side of the subcolumn (FO_SubCol object) containing it.
FP_LocY†	MetricT	Offset of the top of a character from the top of the subcolumn (FO_SubCol object) containing it.

Property	Type	Meaning
FP_Position	IntT	Position of the text: FV_POS_SUPER: Superscript FV_POS_NORM: Baseline FV_POS_SUB: Subscript
FP_PairKern	IntT	True if Pair Kern is enabled.
FP_Overline	IntT	True if Overline is enabled.
FP_SepOverride	F_ObjHandleT	Color for separation override (FO_Color ID).
FP_Spread	MetricT	Obsolete property, but still functional. See corresponding "tracking" property below.
FP_Stretch	MetricT	Character stretch (set width) expressed as a percentage of normal stretch for the font (metric –10% to 1000%).
FP_Strikethrough	IntT	True if Strikethrough is enabled.
FP_StyleOverrides	IntT	Bit flags that specify which overrides (condition indicators) are used: FV_CS_CHANGEBAR FV_CS_NO_OVERRIDE FV_CS_OVERLINE FV_CS_STRIKETHROUGH FV_CS_SINGLE_UNDERLINE FV_CS_DOUBLE_UNDERLINE
FP_Tracking	MetricT	Character tracking expressed as a percentage of an em (metric –100% to 1000%).
FP_Underlining	IntT	Type of underlining: FV_CB_NO_UNDERLINE FV_CB_SINGLE_UNDERLINE FV_CB_DOUBLE_UNDERLINE FV_CB_NUMERIC_UNDERLINE

Property	Type	Meaning
`FP_UseSepOverride`	`IntT`	`True` if color specified for `FP_SepOverride` overrides default.
`FP_Width`†	`MetricT`	Width of a character

a. In the API, most percentages are represented as `MetricT` fractions. For spread percentages, the `MetricT` value `1<<16` or `0x10000` specifies 100% or 1. For more information on `MetricT` values, see "MetricT values" on page 954.

Variables

The API uses an `FO_Var` object to represent a variable instance and an `FO_VarFmt` object to represent a variable format.

FO_Var

`FO_Var` objects have the following properties.

Property	Type	Meaning
`FP_Element`†	`F_ObjHandleT`	If the variable is in a Structured FrameMaker document, the ID of the element associated with the variable.
`FP_Locked`	`IntT`	`True` if the variable is included in a text inset that gets its formatting from the source document. The variable is not affected by global formatting performed on the document.
`FP_NextVarInDoc`†	`F_ObjHandleT`	Next variable instance (`FO_Var` ID) in the document.
`FP_TextRange`†	`F_TextRangeT`	The text range the variable instance encompasses.
`FP_VarFmt`	`F_ObjHandleT`	The variable instance's format (`FO_VarFmt` ID).
`FP_Unique`†	`IntT`	The variable's UID.

FO_VarFmt

`FO_VarFmt` objects have the following properties.

Property	Type	Meaning
`FP_Fmt`	`StringT`	The variable format definition; the building blocks and text strings used to create a variable instance with the variable format.
`FP_Name`	`StringT`	The variable format's name.

Property	Type	Meaning
`FP_NextVarFmtInDoc`†	`F_ObjHandleT`	Next variable format (`FO_VarFmt` ID) in the document's list of variable formats.

Property	Type	Meaning
FP_SystemVar†	IntT	The variable format's type. FV_VAR_USER_VARIABLE: a user-defined variable format. The following types specify system variable formats: FV_VAR_CURRENT_PAGE_NUM FV_VAR_PAGE_COUNT FV_VAR_CURRENT_DATE_LONG FV_VAR_CURRENT_DATE_SHORT FV_VAR_MODIFICATION_DATE_LONG FV_VAR_MODIFICATION_DATE_SHORT FV_VAR_CREATION_DATE_LONG FV_VAR_CREATION_DATE_SHORT FV_VAR_FILE_NAME_LONG FV_VAR_FILE_NAME_SHORT FV_VAR_HEADER_FOOTER_1 FV_VAR_HEADER_FOOTER_2 FV_VAR_HEADER_FOOTER_3 FV_VAR_HEADER_FOOTER_4 FV_VAR_HEADER_FOOTER_5 FV_VAR_HEADER_FOOTER_6 FV_VAR_HEADER_FOOTER_7 FV_VAR_HEADER_FOOTER_8 FV_VAR_HEADER_FOOTER_9 FV_VAR_HEADER_FOOTER_10 FV_VAR_HEADER_FOOTER_11 FV_VAR_HEADER_FOOTER_12 FV_VAR_HEADER_FOOTER_13 FV_VAR_HEADER_FOOTER_14 FV_VAR_HEADER_FOOTER_15 FV_VAR_HEADER_FOOTER_16 FV_VAR_HEADER_FOOTER_17 FV_VAR_HEADER_FOOTER_18 FV_VAR_TABLE_CONTINUATION FV_VAR_TABLE_SHEET

Data Types and Structures Reference

5

This chapter describes the data types and structures used in the API.

Data types

The following table lists primitive data types used in the API. For a list of all the data types provided by the FDE, see "Replacing C primitive data types with FDE types" on page 530 of the *FDK Programmer's Guide*.

Frame API data type	Equivalent fundamental type	Size
`BoolT`	`long`	Signed 4 bytes
`ByteT`	`char`	Signed 1 byte
`ErrorT`	`long`	Signed 4 bytes
`ConStringT`	`const unsigned char*`	Pointer
`F_ObjHandleT`	`long`	Unsigned 4 bytes
`FontEncIdT`	`unsigned short`	Unsigned 2 bytes
`GenericT`	`void* or int`	4 bytes
`IntT`	`long`	Signed 4 bytes
`MetricT`	`long`	Signed 4 bytes
`ShortT`	`short`	Signed 2 bytes
`StringT`	`unsigned char*`	Pointer
`UByteT`	`unsigned char`	Unsigned 1 byte
`UCharT`	`unsigned char`	Unsigned 1 byte
`UIntT`	`unsigned long`	Unsigned 4 bytes
`UShortT`	`unsigned short`	Unsigned 2 bytes
`VoidT`	`void`	None

MetricT values

The `MetricT` type is a 32-bit fixed-point number. The API and FDE use `MetricT` values to express linear measurements, such as tab offsets and font sizes. They also use `MetricT` values to express percentages and angular degrees when a high degree of accuracy is required. `MetricT` values should not be confused with metric system values, that is, with centimeters and meters.

Using MetricT values for linear measurements

When used for linear measurement, a `MetricT` unit represents a point (1/72 inch). The 16 most significant bits of a `MetricT` value represent the digits before the decimal; the 16 least significant bits represent the digits after the decimal. Therefore, 1 point is expressed as hexadecimal `0x10000` or decimal `65536`.

The following table lists the units of the measurement systems FrameMaker products support and their `MetricT` equivalents.

Measurement unit	MetricT hex value	MetricT decimal value	Relationship to other units
inch	`0x480000`	`4718592`	72 points
centimeter	`0x1c58b1`	`1857713`	2.54 cm per inch
millimeter	`0x02d5ab`	`185771`	25.4 mm per inch
pica	`0x0c0000`	`786432`	12 points
point	`0x10000`	`65536`	1/72 inch
didot	`0x011159`	`69977`	0.01483 inch
cicero	`0x0cd02c`	`839724`	12 didots

The FDK defines the following constants which you may prefer to use:

```
FV_METRIC_INCH
FV_METRIC_CM
FV_METRIC_MM
FV_METRIC_PICA
FV_METRIC_POINT
FV_METRIC_DIDOT
FV_METRIC_CICERO
```

With these constants, you can specify `MetricT` values in your client code in a way that is easy to understand. For example, you can specify 5 inches with `5*FV_METRIC_INCH`.

The FDE provides platform-independent utility functions that make it easy to manipulate `MetricT` values without converting them. For more information on these functions, see “Metric library” on page 551 of the *FDK Programmer’s Guide*.

Using MetricT values for percentages and angles

The API and FDE use `MetricT` values to represent percentages. For some properties, such as the document property `FP_Zoom` and the character format property `FP_Spread`, the decimal `MetricT` value `65536` specifies 100 percent. For other properties, such as the color property `FP_Black`, `65536` specifies 1 percent. For more information on specifying a particular `MetricT` percentage property, see Chapter 3, “Object Reference.”

When `MetricT` values are used to represent angular degrees, 1 degree is expressed as decimal `65536`.

Data structures

The following structures, which are used for some FDE function parameters, are incompletely defined in the FDK header files:

- `ChannelT`
- `DirHandleT`
- `FilePathT`
- `HandleT`
- `HashT`
- `StringListT`

These functions are incompletely defined because FDK clients should not directly manipulate their fields. Their fields are not guaranteed to be the same for different versions of the FDK. You should declare only pointers to the incompletely defined structures. If you attempt to declare a variable with one of these structures as the type, it will cause a compiler error.

The following sections describe data structures used by API functions.

ChannelT

Incompletely-defined structure used to specify a channel. For more information on channels, see “” on page 539 of the *FDK Programmer’s Guide*.

DirHandleT

Incompletely defined structure used to specify a directory handle. For information about getting get a DirHandleT, see "F_FilePathOpenDir()" on page 551

FilePathT

Incompletely defined structure used to specify a filepath.

HandleT

Incompletely defined structure used to specify a memory handle. For more information on allocating memory with memory handles, see "Allocating memory with handles" on page 544 of the *FDK Programmer's Guide*.

HashT

Incompletely defined structure used to specify a hash table. For more information on using hash tables, see "The hash library" on page 549 of the *FDK Programmer's Guide*.

StringListT

Incompletely-defined structure used to specify a string list. For more information on using string lists, see "The string list library" on page 548 of the *FDK Programmer's Guide*.

F_AttributeDefsT

An `F_AttributeDefsT` structure specifies a set of attribute definitions.

```
typedef struct {
   UIntT len;
   F_AttributeDefT *val;
} F_AttributeDefsT;
```

F_AttributeDefT

The `F_AttributeDefT` structure describes a single attribute definition.

```
typedef struct {
   StringT name; /* The attribute name. */
   BoolT required; /* True if the attribute is required. */
   UByteT flags; /* Attr value is read-only, hidden, or neither.
                 ** See following table.
                 */
   IntT attrType; /* The attribute type. See following table. */
   F_StringsT choices;/* Choices if attrType is FV_AT_CHOICES.*/
   F_StringsT defValues;/* The default if attribute is not
                        ** required. If attrType is
                        ** FV_AT_INTS, FV_AT_STRINGS, etc
                        ** the default can have multiple strings.
                        */
   StringT rangeMin; /* The minimum allowed value (if any). */
   StringT rangeMax; /* The maximum allowed value (if any). */
} F_AttributeDefT;
```

The `F_AttributeDefT.flags` field determines whether an attribute is read-only, hidden, of neither.

Value for flags	Means
`FV_AF_READ_ONLY`	The attribute value is read-only
`FV_AF_HIDDEN`	The attribute value is hidden
`NULL`	The attribute value is neither read-only nor hidden

The `F_AttributeDefT.attrType` identifies the attribute value's type. It can specify one of the following constants

attrType constant	Attribute value type
`FV_AT_STRING`	Any arbitrary text string
`FV_AT_STRINGS`	One or more arbitrary text strings
`FV_AT_CHOICES`	A value from a list of choices
`FV_AT_INTEGER`	A signed whole number (optionally restricted to a range of values)
`FV_AT_INTEGERS`	One or more integers (optionally restricted to a range of values)

FV_AT_REAL	A real number (optionally restricted to a range of values)
FV_AT_REALS	One or more real numbers (optionally restricted to a range of values)
FV_AT_UNIQUE_ID	A string that uniquely identifies the element
FV_AT_UNIQUE_IDREF	A reference to a UniqueID attribute
FV_AT_UNIQUE_IDREFS	One or more references to a UniqueID attribute

F_AttributesT

An `F_AttributesT` structure specifies a set of attributes.

```
typedef struct {
   UIntT len;
   F_AttributeT *val;
} F_AttributesT;
```

F_AttributeT

The `F_AttributeT` structure describes a single attribute.

```
typedef struct {
   StringT name; /* The attribute name. */
   F_StringsT values; /* The attribute values. */
   UByteT valflags; /* Validation error flags. */
   UByteT allow; /* Allow error as special case. */
} F_AttributeT;
```

F_CombinedFontsT

`F_CombinedFontsT` provides an array of `F_CombinedFontT` structures. The FDK uses it to specify permutations of font characteristics for a combined font.

```
typedef struct {
   UIntT len;
   F_CombinedFontT *val;
} F_CombinedFontsT;
```

F_CombinedFontT

F_CombinedFontT specifies a set of font characteristics for a combined font. The combinedFont field specifies the ID of a FO_CombinedFontDefn object. From this object you can get information about the combined font such as the base and Western font families, or the combined font name. For the list of properties for a combined font definition, see "FO_CombinedFontDefn" on page 759.

The other fields each specify an index into a list of names in the FrameMaker product session. For example, the weight field specifies the index of a name in the list of names specified by the session property FP_FontWeightNames.

```
typedef struct {
   F_ObjHandelT combinedFont; /* Combined font ID */
   UIntT variation; /* Index of the font variation. */
   UIntT weight; /* Index of the font weight. */
   UIntT angle; /* Index of the font angle. */
} F_CombinedFontT;
```

F_CompareRetT

F_CompareRetT provides the results of a call to F_ApiCompare().

```
typedef struct {
    F_ObjHandleT sumId; /* The ID of the summary document. */
    F_ObjHandleT compId; /* The ID of the composite document. */
} F_CompareRetT;
```

F_ElementLocT

The F_ElementLocT structure specifies a location within a structural element. The parent element is the element that contains the location. The child element is the child of parent that immediately follows the location. The offset counts at what number of characters into parent to set the location.

For example, assume an element named List that can contain elements named Item. To set a location before the first Item in List, parent is the ID of the List element, child is the ID of the first Item element in List, and offset is 0. To set the location after the last Item in List, parent is the ID of the list element, child is 0, and offset is 0.

```
typedef struct {
   F_ObjHandleT parentId; /* Parent element ID. */
   F_ObjHandleT childId; /* Child element ID. */
   IntT offset; /* Offset within child/parent element. */
} F_ElementLocT;
```

F_ElementRangeT

An `F_ElementRangeT` structure specifies an element range.

```
typedef struct {
   F_ElementLocT beg; /* Beginning of the element range. */
   F_ElementLocT end; /* End of the element range. */
} F_ElementRangeT;
```

F_FontEncT

`F_FontEncT` stores information about the character set for a given encoding. The encoding is specified by `name`, and can be one of `FrameRoman`, `JISX0208.ShiftJIS`, `BIG5`, `GB2312-80.EUC`, or `KSC5601-1992`. Each element in the arrays indicates whether its corresponding character code is a part of the font set for the given encoding.

The FDE creates an initial set of `F_FontEncT`s for the session, and includes functions to return an ID for each one. You should not have to inspect this structure directly.

```
typedef struct {
   UCharT firstBytes[256];  /* First byte character information
*/
   UCharT secondBytes[256];  /* Second byte character
information */
   StringT name;  /* The name of the encoding */
} F_FontEncT;
```

F_FontsT

`F_FontsT` provides an array of `FontT` structures. The FDK uses it to specify permutations of font characteristics.

```
typedef struct {
   UIntT len;
   F_FontT *val;
} F_FontsT;
```

F_FontT

`F_FontT` specifies a combination of font characteristics. Each field specifies an index into a list of names in the FrameMaker product session. For example, the `family` field specifies the index of a name in the list of names specified by the session property `FP_FontFamilyNames`. The `weight` field specifies the index of a name in the list of names specified by the session property `FP_FontWeightNames`.

```
typedef struct {
   UIntT family; /* Index of the font family. */
   UIntT variation; /* Index of the font variation. */
   UIntT weight; /* Index of the font weight. */
   UIntT angle; /* Index of the font angle. */
} F_FontT;
```

F_ElementCatalogEntryT

`F_ElementCatalogEntryT` describes a catalog entry in an Element Catalog in Structured FrameMaker

```
typedef struct {
   F_ObjHandleT objId; /* ID of element definition */
   IntT flags; /* Validation type */
} F_ElementCatalogEntryT;
```

The `F_ElementCatalogEntryT.flags` field can be one of the following constants.

Flags constant	Meaning
`FV_STRICTLY_VALID`	Catalog entry is strictly valid
`FV_LOOSELY_VALID`	Catalog entry is loosely valid
`FV_ALTERNATIVE`	Catalog entry is an alternative
`FV_INCLUSION`	Catalog entry is valid because it is an inclusion

If no flags are set, the element is invalid at the current position.

F_ElementCatalogEntriesT

`F_ElementCatalogEntriesT` describes the value of the Structured FrameMaker Element Catalog for the current insertion point or text selection.

```
typedef struct {
   UIntT len; /* Number of elements */
   F_ElementCatalogEntryT *val;
} F_ElementCatalogEntriesT;
```

F_IntsT

`F_IntsT` provides an array of `IntT` values.

```
typedef struct {
   UIntT len; /* Number of IntTs */
   IntT *val; /* Array of IntTs */
} F_IntsT;
```

F_MetricsT

`F_MetricsT` provides a set of metric values.

```
typedef struct {
    UIntT len; /* Number of metric values*/
    MetricT *val; /* Array of metric values */
} F_MetricsT;
```

F_PointT

`F_PointT` describes an individual coordinate pair. FrameMaker products measure coordinates from the upper-left corner of the parent frame.

```
typedef struct {
   MetricT    x,y; /* The coordinate pair */
} F_PointT;
```

F_PointsT

`F_PointsT` describes a set of coordinate pairs. It is normally used to describe the vertices of a graphic object, such as a polyline or polygon.

```
typedef struct {
   UIntT len; /* Number of coordinate pairs */
   F_PointT *val; /* Vector of coordinate pairs */
} F_PointsT;
```

F_PropIdentT

`F_PropIdentT` provides a property identifier. Properties can be identified by either a name or a number (integer constant). The API provides defined constants for property numbers (for example, `FP_Fill` and `FP_Height`). Only inset properties (facets) are identified by names.

If a property is identified by a name, `F_PropIdentT.num` is set to `0`. If a property is identified by a number, `F_PropIdentT.name` is set to a null string.

```
typedef struct {
   IntT num; /* The property number */
   StringT name; /* The property name */
}F_PropIdentT;
```

F_PropValT

`F_PropValT` describes a property-value pair.

```
typedef struct {
   F_PropIdentT propIdent; /* The property identifier */
   F_TypedValT propVal; /* The property value */
} F_PropValT;
```

F_PropValsT

`F_PropValsT` describes a property list (set of property-value pairs).

```
typedef struct {
   UIntT len; /* Number of property-value pairs */
   F_PropValT *val; /* Array of pairs */
} F_PropValsT;
```

F_StringsT

`F_StringsT` specifies a set of strings.

```
typedef struct {
   UIntT len; /* Number of strings */
   StringT *val; /* Array of strings */
} F_StringsT;
```

F_TabT

`F_TabT` describes an individual tab. Note that the character specified by `decimal` must be a single byte character.

```
typedef struct {
   MetricT x; /* Offset from the left margin */
   UCharT type; /* Type of tab (see following table) */
   StringT leader; /* String that appears before the tab */
   UCharT decimal; /* Character to align tab around, e.g. "," */
} F_TabT;
```

The `F_TabT.type` field can contain one of the following constants.

Type constant	Tab type
`FV_TAB_LEFT`	Left tab
`FV_TAB_CENTER`	Center tab
`FV_TAB_RIGHT`	Right tab
`FV_TAB_DECIMAL`	Decimal tab
`FV_TAB_RELATIVE_LEFT`	Relative center tab (allowed only for format change lists)
`FV_TAB_RELATIVE_CENTER`	Relative right tab (allowed only for format change lists)
`FV_TAB_RELATIVE_RIGHT`	Relative decimal tab (allowed only for format change lists)
`FV_TAB_RELATIVE_DECIMAL`	Relative center tab (allowed only for format change lists)

F_TabsT

`F_TabsT` specifies the set of tabs in a paragraph or Paragraph Catalog format.

```
typedef struct {
   UIntT len; /* Number of tabs */
   F_TabT *val; /* Array of tabs */
} F_TabsT;
```

F_TextItemT

`F_TextItemT` describes an individual text item or unit of text within a paragraph or graphic text line. A text item can represent:

- A string of characters with a common character format and condition tag

- The beginning or end of a line, paragraph, flow, column, or page
- An anchor for a table, footnote, marker, cross-reference, variable, or anchored frame
- A text properties change

```
typedef struct {
   IntT offset; /* From paragraph or text line beginning. */
   IntT dataType; /* Text item type (see following table) */
   union {
      StringT sdata; /* String if text item is FTI_String */
      IntT idata; /* ID of object if text item is object */
      } u;
} F_TextItemT;
```

The following table lists the values the `F_TextItemT.dataType` field can have and the types of data the corresponding text item provides.

Text item type (dataType)	What the text item represents	Text item data
`FTI_TextObjId`	The object that the offsets of all the text items are relative to	ID of an `FO_Pgf`, `FO_Cell`, `FO_TextLine`, `FO_TiApiClient`, `FO_TiFlow`, `FO_TiText`, or `FO_TiTextTable`
`FTI_String`	A string of characters with the same condition and character format	A character string
`FTI_LineBegin`	The beginning of a line	Nothing
`FTI_LineEnd`	The end of a line and the line-end type	If the line end is a normal line end, `0`; if it is a forced line end, the `FTI_HardLineEnd` flag is set; if it is a hyphen line end, the `FTI_HyphenLineEnd` flag is set
`FTI_PgfBegin`	The beginning of a paragraph	ID of an `FO_Pgf`
`FTI_PgfEnd`	The end of a paragraph	ID of an `FO_Pgf`
`FTI_FlowBegin`	The beginning of a flow	ID of an `FO_Flow`
`FTI_FlowEnd`	The end of a flow	ID of an `FO_Flow`
`FTI_PageBegin`	The beginning of a page	ID of an `FO_Page`

Text item type (dataType)	What the text item represents	Text item data
`FTI_PageEnd`	The end of a page	ID of an `FO_Page`
`FTI_SubColBegin`	The beginning of a column	ID of an `FO_SubCol`
`FTI_SubColEnd`	The end of a column	ID of an `FO_SubCol`
`FTI_FrameAnchor`	An anchored frame	ID of an `FO_AFrame`
`FTI_FnAnchor`	A footnote	ID of an `FO_Fn`
`FTI_TblAnchor`	A table	ID of an `FO_Tbl`
`FTI_MarkerAnchor`	A marker	ID of an `FO_Marker`
`FTI_XRefBegin`	The beginning of a cross-reference	ID of an `FO_XRef`
`FTI_XRefEnd`	The end of a cross-reference	ID of an `FO_XRef`
`FTI_TextFrameBegin`	The beginning of a text frame	ID of an `FO_TextFrame`
`FTI_TextFrameEnd`	The end of a text frame	ID of an `FO_TextFrame`
`FTI_VarBegin`	The beginning of a variable	ID of an `FO_Var`
`FTI_VarEnd`	The end of a variable	ID of an `FO_Var`
`FTI_ElementBegin`	The beginning of a structural container element	ID of an `FO_Element`
`FTI_ElementEnd`	The end of a structural container element	ID of an `FO_Element`
`FTI_ElemPrefixBegin`	The beginning of an element's prefix	ID of an `FO_Element`
`FTI_ElemPrefixEnd`	The end of an element's prefix	ID of an `FO_Element`
`FTI_ElemSuffixBegin`	The beginning of an element's suffix	ID of an `FO_Element`
`FTI_ElemSuffixEnd`	The end of an element's suffix	ID of an `FO_Element`
`FTI_CharPropsChange`	A change in the text properties	Flags indicating which properties have changed (see the table below)
`FTI_RubiCompositeBegin`	The beginning of a rubi composite (and the beginning of oyamoji text).	ID of an `FO_Rubi`
`FTI_RubiCompositeEnd`	The end of a rubi composite.	ID of an `FO_Rubi`

Text item type (dataType)	What the text item represents	Text item data
FTI_RubiTextBegin	The beginning of rubi text (and the end of oyamoji text).	ID of an FO_Rubi
FTI_RubiTextEnd	The end of rubi text.	ID of an FO_Rubi

The following table lists the bit flags that a client can bitwise AND with the idata field of an FTI_CharPropsChange text item. For example, to determine if the font family changed, bitwise AND the FTF_FAMILY flag with the idata field.

Flag	Meaning
FTF_FAMILY	The font family has changed.
FTF_VARIATION	The font variation has changed.
FTF_WEIGHT	The font weight has changed.
FTF_ANGLE	The font angle has changed.
FTF_UNDERLINING	The underlining has changed.
FTF_STRIKETHROUGH	The strikethrough characteristic has changed.
FTF_OVERLINE	The overline characteristic has changed.
FTF_CHANGEBAR	The change bars have changed.
FTF_OUTLINE	The outline characteristic has changed.
FTF_SHADOW	The shadow characteristic has changed.
FTF_PAIRKERN	The pair kerning has changed.
FTF_SIZE	The font size has changed.
FTF_KERNX	The kern-x characteristic has changed.
FTF_KERNY	The kern-y characteristic has changed.
FTF_SPREAD	The font spread has changed.
FTF_COLOR	The color has changed.
FTF_CHARTAG	The Character Catalog format has changed.
FTF_CAPITALIZATION	The capitalization has changed.
FTF_POSITION	The character position has changed.
FTF_CONDITIONTAG	The condition tag has changed.
FTF_STRETCH	Font stretch value has changed

Flag	Meaning
FTF_LANGUAGE	Character language has changed
FTF_TSUME	Tsume setting has changed
FTF_IIF	Inline input has changed
FTF_ENCODING	The text encoding has changed.
FTF_ALL	OR of all the flags listed above.

F_TextItemsT

F_TextItemsT provides the text items in a paragraph or a graphic text line.

```
typedef struct {
   UIntT len; /* Number of text items */
   F_TextItemT *val; /* Array of text items */
} F_TextItemsT;
```

F_TextLocT

F_TextLocT specifies a location within the text of a paragraph or a graphic text line.

```
typedef struct{
   F_ObjHandleT objId; /* FO_Pgf or FO_TextLine */
   IntT offset; /* Characters from beginning */
} F_TextLocT;
```

F_TextRangeT

F_TextRangeT specifies a text range. A text range can span paragraphs. It can't span graphic text lines or flows. Note that beg.offset and end.offset fields of a F_TextRangeT can specify offsets relative to the beginning of an object, but they can also use the special value FV_OBJ_END_OFFSET. FV_OBJ_END_OFFSET specifies the offset of the last character in the object containing the text range.

```
typedef struct {
   F_TextLocT beg; /* The beginning of the range */
   F_TextLocT end; /* The end of the range */
} F_TextRangeT;
```

F_TypedValT

`F_TypedValT` specifies an individual property value.

```
typedef struct {
   IntT valType; /* The type of value. See following table. */
   union {
      StringT sval; /* String value */
      F_StringsT ssval; /* Set of strings */
      F_MetricsT msval; /* Set of metrics */
      F_PointsT psval; /* Set of points */
      F_TabsT tsval; /* Set of tabs */
      F_TextLocT tlval; /* Text location */
      F_TextRangeT trval; /* Text range */
      F_AttributeDefsT adsval;
      F_AttributesT asval;
      F_ElementCatalogEntriesT csval; /* Element Catalog */
      F_IntsT isval; /* Set of integers */
      F_UIntsT uisval; /* Set of unsigned integers */
      IntT ival; /* integer */
   } u;
} F_TypedValT;
```

The `F_TypedValT.valType` field indicates the type of value the structure provides—that is, which field of the `u` union is used. The constants used in the `valType` field are described in the following table.

valType constant	Property data type	u field
FT_AttributeDefs	F_AttributeDefsT	adsval
FT_Attributes	F_AttributesT	asval
FT_Integer	IntT	ival
FT_Ints	F_IntsT	isval
FT_Metric	MetricT	ival
FT_Metrics	F_MetricsT	msval
FT_String	StringT	sval
FT_Strings	F_StringsT	ssval
FT_Id	F_ObjHandleT	ival
FT_Points	F_PointsT	psval
FT_Tabs	F_TabsT	tsval

valType constant	Property data type	u field
FT_TextLoc	F_TextLocT	tlval
FT_TextRange	F_TextRangeT	trval
FT_UInts	F_UIntsT	uisval
FT_ElementCatalog	F_ElementCatalogEntriesT	csval
FT_UBytes	F_UBytesT	No field

IMPORTANT: *Integer (*IntT*), metric (*MetricT*), and ID (*F_ObjHandleT*) values are all put in the* ival *field of the* u *union.*

F_UBytesT

F_UBytesT provides an array of unsigned byte (UbyteT) values. It is used to provide facet data for insets.

```
typedef struct {
    UIntT len; /* Number of unsigned bytes */
    UByteT *val; /* Array of unsigned bytes */
} F_UBytesT;
```

F_UIntsT

F_UIntsT provides an array of UIntT values.

```
typedef struct {
   UIntT len; /* Number of UIntTs */
   UIntT *val; /* Array of UIntTs */
} F_UIntsT;
```

6

Error Codes

If the API encounters an error condition, it assigns an error code to the global variable `FA_errno`. The following table lists the error codes and their meanings. Error codes are also listed in the `fapidefs.h` header file provided with the FDK.

Error code	Meaning
`FE_AsianSystemRequired`	The API expects an Asian operating system
`FE_BadBaseColor`	The client specified a tint as a base color, or specified the actual color as a base color
`FE_BadBookId`	The client specified an invalid book ID.
`FE_BadCompare`	The document comparison failed.
`FE_BadCompPath`	The client specified an invalid path for a new book component.
`FE_BadDelete`	The API can't delete the specified object.
`FE_BadDocId`	The client specified an invalid document ID.
`FE_BadElementDefId`	The API expected the ID of an element definition (`FO_ElementDef`).
`FE_BadElementId`	The API expected the ID of an element (`FO_Element`).
`FE_BadElementSelection`	Invalid element range specified for `F_ApiSetElementSelection()`.
`FE_BadFamilyName`	The client specified a name of a color library that doesn't exist
`FE_BadFileType`	The file type expected by the API does not match the type of file being imported or the type of the source file for the text inset being updated.
`FE_BadInkName`	The specified color library does not have an ink of this name
`FE_BadInsertPos`	The client specified an invalid insertion position for text or a new object.

Error code	Meaning
FE_BadMenuBar	The API expected the menu (FO_Menu) to contain menus (FO_Menu) only.
FE_BadName	The application tried to name an object with an illegal name.
FE_BadNew	There was an illegal call to a create a new object.
FE_BadNewFrame	The API can't move the specified object to the specified frame.
FE_BadNewGroup	The API can't move the specified object to the specified graphic object group (FO_Group).
FE_BadNewSibling	The API can't make an object a sibling of the specified object.
FE_BadNotificationNum	The application called F_ApiNotification() with an invalid notification constant.
FE_BadObjId	The application specified an invalid object ID.
FE_BadOperation	The API tried to execute an illegal operation.
FE_BadPageDelete	The API can't delete the specified page.
FE_BadParameter	The application attempted to call an API function with an invalid parameter.
FE_BadPropNum	The application attempted to get or set a property that the specified object doesn't have.
FE_BadPropType	The application attempted to get or set a property with the wrong function.
FE_BadRange	The specified text range is invalid (for example, because two ends of a range are not in the same flow or are hidden).
FE_BadSaveFileName	The API tried to save a file with an illegal name.
FE_BadSelectionForOperation	The document selection is not valid for the operation.
FE_BadShortcut	The keyboard shortcut is invalid.
FE_BookUnStructured	Application called F_ApiNewBookComponentInHierarchy() and the specified book was unstructured.

Error code	Meaning
FE_Busy	The FrameMaker product is in a busy state and cannot process the API request (for example, because a drag operation is in progress or because a modal dialog box is currently displayed). This error code is only returned to Windows asynchronous clients (for example, clients initiated by DDE callbacks or timer procedures).
FE_Canceled	An operation (such as Save or Open) was canceled.
FE_CanceledByClient	An FDK client cancelled the operation.
FE_CantSmooth	The API can't smooth or unsmooth the specified object.
FE_CantUpdateMacEdition	The specified text inset is a Macintosh edition, which can't be updated on the current, non-Macintosh platform.
FE_CircularReference	Importing the document would cause a circular reference.
FE_CompareTypes	The API attempted to compare files of different types (for example, a book and a document).
FE_DocModified	The application attempted to close a modified document without specifying FF_CLOSE_MODIFIED.
FE_DupName	A same-type item with this name exists.
FE_EmptyTextObject	The object contains no text.
FE_FailedState	The API tried to access something that is missing; for example, a dialog box or an object ID.
FE_FileClosedByClient	The file was closed by an API client when it processed a notification.
FE_FilterFailed	The filter failed to import or export a file
FE_GenRuleAmbiguous	A general rule in structured document is ambiguous.
FE_GenRuleConnectorExpected	A general rule in structured document is missing a connector.
FE_GenRuleItemExpected	A general rule in structured document is missing a rule item.
FE_GenRuleLeftBracketExpected	A general rule in structured document is missing a left bracket.

Error code	Meaning
FE_GenRuleMixedConnectors	A general rule in structured document has mixed connectors.
FE_GenRuleRightBracketExpected	A general rule in structured document is missing a right bracket.
FE_GenRuleSyntaxError	A general rule in structured document has a syntax error.
FE_GroupSelect	The API can't select or deselect an object in the specified group.
FE_HiddenPage	The value must be the ID of a page that is not hidden.
FE_InvalidString	Specification has syntax error.
FE_InvAttribute	The F_AttributeT value is invalid.
FE_InvAttributeDef	The F_AttributeDefT value is invalid.
FE_MissingFile	The specified file no longer exists.
FE_NameNotFound	The API can't find the object with the specified name.
FE_NoColorFamily	The client tried to set an ink name without specifying a color library
FE_NoSuchFlow	The specified flow does not exist in the source document.
FE_NotApiCommand	The API expected the ID of a command defined by an API client (FO_Command).
FE_NotBodyPage	The API expected the ID of a body page (FO_BodyPage).
FE_NotBookComponent	The API expected a book component.
FE_NotCommand	The API expected the ID of a command (FO_Command).
FE_NotFound	F_ApiFind() did not find an occurance of the searched item
FE_NotFrame	The value must be the ID of a frame (FO_Frame).
FE_NotGraphic	The value must be the ID of a graphic object.
FE_NotGroup	The value must be the ID of a graphic object group (FO_Group).
FE_NotInMenu	The specified menu item (FO_Command or FO_Menu) is not in the menu.

Error code	Meaning
FE_NotMenu	The API expected the ID of a menu (FO_Menu).
FE_NotPgf	The API expected the ID of a paragraph (FO_Pgf).
FE_NotPgfOrFlow	The API expected the ID of a paragraph (FO_Pgf) or text flow (FO_Flow).
FE_NotTextFrame	The API expected the ID of a text frame (FO_TextFrame).
FE_NotTextObject	The value must be the ID of a text object (FO_Pgf, FO_TextLine, FO_Flow, FO_Cell, FO_TextFrame, FO_SubCol, FO_Fn, FO_Element, FO_XRef, FO_Var, FO_TiFlow, FO_TiText, FO_TiTextTable, FO_TiApiClient).
FE_OffsetNotFound	The specified offset couldn't be found in the specified paragraph or text line.
FE_OutOfRange	The specified value is not in the legal range for the specified property.
FE_PageFrame	The page frame can't be moved or selected.
FE_PropNotSet	Specified property is not set in the Format Change List
FE_ReadOnly	The application attempted to set a read-only property.
FE_ReservedColor	The client tried to change a fixed property of a reserved color
FE_SomeUnresolved	Some cross-references are unresolved.
FE_Success	The API function call was successful.
FE_SystemError	Unable to open the document due to system error.
FE_TintedColor	The client tried to change a tinted color in a way that is inappropriate for the tint's base color
FE_Transport	Communication between the FrameMaker product and the API application is not working correctly.
FE_TypeUnNamed	The API can't use F_ApiGetNamedObject() or F_ApiGetUnique() for the specified object type.
FE_ViewOnly	The API tried to modify a View Only document.
FE_WantsCustom	The user clicked Custom in the New dialog box.

Error code	Meaning
FE_WantsLandscape	The user clicked Landscape in the New dialog box.
FE_WantsPortrait	The user clicked Portrait in the New dialog box.
FE_WithinFrame	Must move between frames before the requested operation is possible.
FE_WrongProduct	The current FrameMaker product doesn't support the requested operation.

Calling Clients Shipped with FrameMaker

7

A number of features in FrameMaker are implemented via FDK clients. For example, table sorting and importing XML are implemented in this way. This chapter describes how you can use `F_ApiCallClient()` to call various FDK clients that are shipped with FrameMaker.

Calls to work with structured documents

Much of the structured document functionality FrameMaker provides is implemented via FDK clients. To call this functionality programmatically, you use `F_ApiCallClient()`. The following table lists some FrameMaker functionality and the registered names of the clients you pass to `F_ApiCallClient()` to invoke it programmatically.

Functionality	Registered client name
Element catalog manager	`Element Catalog Manager`
Structure generator	`Structure Generator`
Reading and writing markup documents and reading, writing, and updating DTD and EDD documents	`FmDispatcher`

To execute most operations with FrameMaker clients, you must make a sequence of several `F_ApiCallClient()` calls. In each call, you set `clname` to the registered name of the client and arg to a special string value. The following tables list the sequence of calls for each operation.

To structure a document, use the following sequence of `F_ApiCallClient()` calls:

F_ApiCallClient() calls to structure a document
`F_ApiCallClient("Structure Generator", "InputDocId` *objectID*`");` where *objectID* is the ID of the input document

`F_ApiCallClient("Structure Generator", "RuleDocId` *objectID*");
where *objectID* is the ID of the rule table document.

`F_ApiCallClient("Structure Generator", "OutputDocName` *pathname*"); where *pathname* is the full pathname of the output document. This command is optional. If you do not specify a pathname, the structure generator leaves the document unsaved and open.

`F_ApiCallClient("Structure Generator", "LogName` *pathname*"); where *pathname* is the full pathname of a log file. If you do not specify a pathname, the structure generator creates a log, but leaves the log file unsaved and open.

`F_ApiCallClient("Structure Generator", "StructureDoc");` This call instructs the structure generator to generate structure, using the parameters provided by the calls listed above.

To structure a book, use the following sequence of `F_ApiCallClient()` calls:

F_ApiCallClient() calls to structure a book

`F_ApiCallClient("Structure Generator", "InputBookId` *objectID*"); where *objectID* is the ID of the input book.

`F_ApiCallClient("Structure Generator", "RuleDocId` *objectID*"); where *objectID* is the ID of the rule table document.

`F_ApiCallClient("Structure Generator", "OutputDirName` *dirpath*"); where *dirpath* is the full pathname of the directory in which you want FrameMaker to save the structured documents. This command is optional. If you do not specify a pathname, the structure generator leaves the documents unsaved and open.

`F_ApiCallClient("Structure Generator", "LogName` *pathname*"); where *pathname* is the full pathname of a log file. This command is optional. If you do not specify a pathname, the structure generator leaves the log file unsaved and open.

`F_ApiCallClient("Structure Generator", "GenerateBook");`
This call instructs the structure generator to generate structure, using the parameters provided by the calls listed above.

To generate a conversion table, use the following sequence of `F_ApiCallClient()` calls:

F_ApiCallClient() calls to generate a conversion table

`F_ApiCallClient("Structure Generator", "SourceDocId` *docId*);
where *docID* is the ID of the document for which to generate the conversion table.

`F_ApiCallClient("Structure Generator", "GenerateTable");`
This call instructs the structure generator to generate the conversion table, using the parameters provided by the call above.

To update a conversion table, use the following sequence of `F_ApiCallClient()` calls:

F_ApiCallClient() calls to update a conversion table

`F_ApiCallClient("Structure Generator", "SourceDocId` *docId*);
where *docID* is the ID of the document with which to update the conversion table.

`F_ApiCallClient("Structure Generator", "UpdateTable` *TableID*");
where *TableId* is the ID of the conversion table document. This call instructs the structure generator to update the conversion table, using the parameters provided by the call above.

To initialize FmDispatcher, use the following sequence of `F_ApiCallClient()` calls:

F_ApiCallClient() calls to initialize FmDispatcher

`F_ApiCallClient("FmDispatcher", "[SgmlInit | XmlInit]");`
This call begins initialization of FmDispatcher for import/export of SGML or XML files. Note that you need to initialize FmDispatcher separately for each type of markup you want to import or export; SGML or XML.

`F_ApiCallClient("FmDispatcher", "StructuredApps` pathname");
where pathname is the full pathname of the sgmlapps file to read. This command is optional. If you do not use it, FmDispatcher reads structapps.fm. For compatibility with earlier FDK clients, FmDispatcher still recognizes `SgmlApps`.

F_ApiCallClient() calls to initialize FmDispatcher

`F_ApiCallClient("FmDispatcher", "StructuredNoApp [SGML | XML]");`

or

`F_ApiCallClient("FmDispatcher", "StructuredAppName` *appName*");

where *appname* is the name of the structure application to use. If you call `FmDispatcher` with `StructuredNoApp`, you should specify either SGML or XML. This determines whether `FmDispatcher` runs using the SGML parser or the XML parser. If you don't specify SGML or XML, it assumes SGML by default.

For compatibility with earlier FDK clients, FmDispatcher still recognizes `SgmlNoApp` and `SgmlAppName`.

`F_ApiCallClient("FmDispatcher", "StructuredOutputDir` *dirName*");

where *dirName* is the full pathname of the directory in which the client saves the filtered file. For compatibility with earlier FDK clients, `FmDispatcher` still recognizes `SgmlOutputDir`.

`F_ApiCallClient("FmDispatcher", "StructuredOutputSuffix` *suffix*");
where *suffix* is the suffix appended to the names of the filtered files. If you do not specify a suffix, the client uses `sgm` for SGML files,, `xml` for XML files, `fm` for FrameMaker product files, `edd` for EDD files, and `dtd` for DTD files. For compatibility with earlier FDK clients, `FmDispatcher` still recognizes `SgmlOutputSuffix`.

`F_ApiCallClient("FmDispatcher", "StructuredLog` *pathname*");

where *pathname* is the full pathname of a log file. For compatibility with earlier FDK clients, `FmDispatcher` still recognizes `SgmlLog`.

`F_ApiCallClient("FmDispatcher", "StructuredFlow` *flowTag*");

where *flowtag* is the tag of the flow to filter, if you are filtering a FrameMaker product document to markup. For compatibility with earlier FDK clients, `FmDispatcher` still recognizes `SgmlFlow`

To translate FrameMaker product documents to SGML or XML files, use the following sequence of `F_ApiCallClient()` calls:

F_ApiCallClient() calls to translate FrameMaker documents to SGML or XML files

Calls to initialize FmDispatcher. See "F_ApiCallClient() calls to initialize FmDispatcher" on page 979. Be sure to initialize FmDispatcher for SGML or XML, as appropriate.

`F_ApiCallClient("FmDispatcher", "StructuredStartList");`
This call starts the list of input files. For compatibility with earlier FDK clients, FmDispatcher still recognizes `SgmlStartList`.

`F_ApiCallClient("FmDispatcher", "StructuredWriteSgml` *pathname*");
or
`F_ApiCallClient("FmDispatcher", "StructuredWriteXml pathname");`
where *pathname* is the full pathname of the FrameMaker product file to translate. Repeat this call for each FrameMaker product file you want to translate. For compatibility with earlier FDK clients, FmDispatcher still recognizes `SgmlWriteSgml`.

`F_ApiCallClient("FmDispatcher", "StructuredEndList");`
This call ends the list of input files and starts file translation. For compatibility with earlier FDK clients, `FmDispatcher` still recognizes `SgmlEndList`.

To translate SGML or XML files to FrameMaker product documents, use the following sequence of `F_ApiCallClient()` calls:

F_ApiCallClient() calls to translate SGML or XML files to FrameMaker documents

Calls to initialize FmDispatcher. See "F_ApiCallClient() calls to initialize FmDispatcher" on page 979. Be sure to initialize FmDispatcher for SGML or XML, as appropriate.

`F_ApiCallClient("FmDispatcher", "StructuredStartList");`
This call starts the list of input files. For compatibility with earlier FDK clients, FmDispatcher still recognizes `SgmlStartList`.

F_ApiCallClient() calls to translate SGML or XML files to FrameMaker documents

`F_ApiCallClient("FmDispatcher", "StructuredReadSgml` *pathname*");

or

`F_ApiCallClient("FmDispatcher", "StructuredReadXml pathname");`

where *pathname* is the full pathname of the FrameMaker product file to translate. Repeat this call for each markup file you want to translate. For compatibility with earlier FDK clients, FmDispatcher still recognizes `SgmlWriteSgml`.

`F_ApiCallClient("FmDispatcher", "StructuredEndList");`

This call ends the list of input files and starts file translation. For compatibility with earlier FDK clients, `FmDispatcher` still recognizes `SgmlEndList`.

To translate a DTD file to an EDD file, use the following sequence of `F_ApiCallClient()` calls:

F_ApiCallClient() calls to translate a DTD file to an EDD file

Calls to initialize FmDispatcher. See "F_ApiCallClient() calls to initialize FmDispatcher" on page 979. Be sure to initialize FmDispatcher for SGML or XML, as appropriate.

`F_ApiCallClient("FmDispatcher", "StructuredUpdateEDD` *pathname*");where *pathname* is the full pathname of an existing EDD file. This call is optional. If you use it, `FmDispatcher` updates the existing EDD file instead of creating a new file. For compatibility with earlier FDK clients, `FmDispatcher` still recognizes `SgmlUpdateEDD`.

`F_ApiCallClient("FmDispatcher", "StructuredReadDtd` *pathname*");

where *pathname* is the full pathname of the DTD file you want to translate. For compatibility with earlier FDK clients, `FmDispatcher` still recognizes `SgmlReadDtd`.

To translate an EDD file to a DTD file, use the following sequence of `F_ApiCallClient()` calls:

F_ApiCallClient() calls to translate an EDD file to an DTD file

Calls to initialize FmDispatcher. See "F_ApiCallClient() calls to initialize FmDispatcher" on page 979. Be sure to initialize FmDispatcher for SGML or XML, as appropriate.

`F_ApiCallClient("FmDispatcher", "StructuredWriteDtd` *pathname*");

where *pathname* is the full pathname of the DTD file you want to translate. For compatibility with earlier FDK clients, `FmDispatcher` still recognizes `SgmlReadDtd`.

To export an EDD file from a structured FrameMaker product document or book, use the following `F_ApiCallClient()` call:

F_ApiCallClient() call to generate an EDD file from a document or book

`F_ApiCallClient("Element Catalog Manager", "ExportEDD` *objectId*");
where *objectId* is the ID of the document or book from which you want to generate and EDD.

To create an EDD file, use the following `F_ApiCallClient`() call:

F_ApiCallClient() call to create an EDD file

```
F_ApiCallClient("Element Catalog Manager", "NewEDD");
```
This call creates a new, unnamed EDD.

To validate an XML document, use the following `F_ApiCallClient()` call:

F_ApiCallClient() call to validate an XML document

`F_ApiCallClient("FmValidateXml", "XML` *pathname*");
where *pathname* is the full pathname of the XML file you want to validate.

Calls to Sort Tables

To sort a table, you first select the rows you want to sort, then use the following `F_ApiCallClient()` calls:

F_ApiCallClient() call to sort a table

```
F_ApiCallClient("TableSort",
"sort_by case index1 order1 index2 order2 index3 order3");
```

where:

- *sort_by* is one of `Row` or `Column`.
- *case* is one of `Case` or `NoCase`.
- *index1* is an integer (starting from 0) to determine the first sort index. If sorting by rows, the column to use as the first sort index; if sorting by columns, the row to use as the first sort index. To sort a table, this value must be greater than `-1`.
- *order1* is one of `Ascending` or `Descending`.
- *index2* and *order2* specify the second sort index. A value of -1 for index indicates no second sort index.
- *index3* and *order3* specify the third sort index. A value of -1 for index indicates no third sort index.

For example,

```
F_ApiCallClient("TableSort", "row nocase 0
descending 2 ascending -1 ascending");
```

tells TableSort to sort by rows, ignoring case. The left most column is the first index (descending), the third column from the left is the second index (ascending), and there is no third index.

The user can specify options for PDFSize via the Optimization Options dialog box, which persist across product sessions. When you use F_ApiCallClient() to run PDFSize, you can use the stored options, or you can specify option settings of your own. You must also specify the file or files to process.

You can choose one of two ways to specify option settings and files to process:

- Pass a pathname to a script file that sets the options and specifies the files to process
- Pass a string that sets the options and specifies the files to process

To optimize PDF file size using options specified in a script file, use the following call:

F_ApiCallClient() call to optimize PDF size, passing a pathname to an options script

```
F_ApiCallClient("PDFSize",
"script=filename");
```

where *filename* is a string for the absolute path to a script file.

The script file includes keywords and values to specify options to use when optimizing PDF file size. These keywords/value pairs correspond to the settings in the Optimization Options dialog box; the `FILE` keyword specifies which documents to process. For any options you do not specify, PDFSize will use the setting last made by the user. All file names must be absolute paths, and must be platform specific.

The keywords and values you can specify in a script are:

- `Interactive = Yes|No`
- `OptimizeAll = Yes|No`
- `ForceOptimization = Yes|No`
- `ClearOptimizationInfo = Yes|No`
- `PromptWhenOpening = Yes|No`
- `PromptWhenSaving = All|Optimized|No`
- `SaveDirectory = directory name`
- `CancelOnError = Yes|No`
- `WarnUnoptimized = Yes|No`
- `LogFile = file name`
- `TraceMode = No|Error|Warning|Info`
- `File = file name` (You can specify any number of files to process)

To optimize PDF file size using options specified in a string, use the following call:

F_ApiCallClient() call to optimize PDF size, passing a string of options

```
F_ApiCallClient("PGFSize",
"string");
```

where *string* is a string that does not begin with the keyword `script`.

The string can include the same keywords and values you can specify for a script; each keyword/value pair is separated by a newline character. For example, the string, "`interactive=No\nFile=usr\myDocs\myFile.fm`" turns off interactive processing and optimizes the file `myFile.fm`.

Calls for WebDAV workgroup management

The FrameMaker product includes support for WebDAV workgroup management, which includes connecting to WebDAV servers, checking files in and out, and updating files. This support is provided by an FDK client named WebDAV.

You execute WebDAV commands via `F_ApiCallClient()` calls. For each call to the WebDAV client you pass a single string, but that string can contain a number of arguments. Each argument in the string is separated by a NULL character. Using this method to pass arguments avoids problems that might arise with spaces in an argument, and it saves you from having to surround arguments in quotes.

Technically speaking, because of the NULL characters the passed string is really a group of strings. But you store these strings in a single character buffer and pass that buffer to the client. You must allocate a buffer that has room for your arguments, plus the NULL characters that separate them, plus a terminating NULL character. Each passed string includes a command name, and if necessary one or more arguments to the command

To define a server mount point or delete a mount point from the current list of definitions, use the following calls. To set up a new server mount point, you provide five NULL-separated command arguments. If you omit a command argument you must still include the NULL separator. To execute the `SetServer` command you must at least provide a nickname, URL, and path. To connect to a currently defined mount point, provide an existing nickname. If an existing mount point is set up without the user name and password in its definition, the user may be prompted for this information for each server access.

The commands to set and delete a WebDAV server are:

F_ApiCallClient() calls to set and delete a WebDAV server

F_ApiCallClient("WebDAV", "SetServer\0nickname\0URL\0name\0password\0path");
where the arguments are listed below, with its corresponding field in the Server Setup dialog box:

- *nickname*: the Server Nickname field
- *URL*: the Server URL field
- *name*: the User Name field
- *password*: the Password field
- *path*: the Workgroup Files Location field

`F_ApiCallClient("WebDAV", "DeleteServer\0`*nickname*`");`

To perform WebDAV actions on files use the following calls, where *filename* specifies a FrameMaker document or book, or else a file created in some other application. In all these calls, *filename* is a string expressing a path in the syntax for the current platform, and *URL* is a string describing the URL to the file on the server.

F_ApiCallClient() calls to perform file actions

`F_ApiCallClient("WebDAV", "SaveToServer\0`*filename*");
Corresponds to the Workgroup > Save and Workgroup > Save As commands.

`F_ApiCallClient("WebDAV", "Update\0`*filename*`");`
Corresponds to the Workgroup > Update command.

`F_ApiCallClient("WebDAV", "CheckOut\0`*filename*`");`
Corresponds to the Workgroup > Check Out command.

`F_ApiCallClient("WebDAV", "CheckIn\0`*filename*`");`
Corresponds to the Workgroup > SaveCheck In command.

`F_ApiCallClient("WebDAV", "CancelCheckOut\0`*filename*`");`
Corresponds to the Workgroup > Cancel Check Out command.

`F_ApiCallClient("WebDAV",`
`"SetDocManagementInfo\0`*filename*`\0`*URL*`\0`*Setting*`");`
where *Setting* is one of `CheckedIn` or `CheckedOut`.

F_ApiCallClient() calls to perform file actions

`F_ApiCallClient("WebDAV", "RemoveDocManagementInfo\0`*filename*`");`
This command clears all document management information from the current document.

`F_ApiCallClient("WebDAV", "GetFileFromServer\0`*URL*`");`
This command corresponds to the Workgroup > Get File From Server command.

`F_ApiCallClient("WebDAV", "PutFileOnServer\0`*filename*`\0`*URL*`");`
This command corresponds to the Workgroup > Put File On Server command.

To manage links to the currently active document, use the following calls which correspond to the menu of the Links palette:

F_ApiCallClient() calls to perform file actions

```
F_ApiCallClient("WebDAV", "SaveAllLinks");
```

```
F_ApiCallClient("WebDAV", "UpdateAllLinks");
```

```
F_ApiCallClient("WebDAV", "CheckInAllLinks");
```

```
F_ApiCallClient("WebDAV", "CheckOutAllLinks");
```

```
F_ApiCallClient("WebDAV", "CancelAllLinkCheckout");
```

To specify workgroup management preferences, use the following calls which correspond to settings in the Workgroup Preferences dialog box. Each command can take one of `Always`, `Ask`, or `Never`, or one of `True` or `False` as its argument.

F_ApiCallClient() calls to set preferences

```
F_ApiCallClient("WebDAV", "UpdateWhenOpening\0[Always | Ask |
Never]");
```

```
F_ApiCallClient("WebDAV", "CheckOutWhenOpening\0[Always | Ask |
Never]");
```

```
F_ApiCallClient("WebDAV","UpdateLinkWhenDocOpened\0[Always | Ask |
Never]");
```

```
F_ApiCallClient("WebDAV",
"CheckInLinkWhenDocCheckedIn\0[Always | Ask | Never]");
```

F_ApiCallClient() calls to set preferences

```
F_ApiCallClient("WebDAV",
"CheckInLinks\0[Always | Ask | Never]");
```

```
F_ApiCallClient("WebDAV",
"UpdateHypertextLinks\0[Always | Ask | Never]");
```

```
F_ApiCallClient("WebDAV",
"EnableWebDAV\0[True | False]");
```

To specify how FrameMaker handles error conditions fo workgroup commands, use the following call. The command can take one of `DoCancel`, `DoShowDialog`, or `DoOk` as its argument.

F_ApiCallClient() calls to set error handling

```
F_ApiCallClient("WebDAV",
"SetInteraction\0[DoCancel | DoShowDialog | DoOk]");
```

To perform an action equivalent to the user choosing a workgroup command from a menu, use the following calls. For these calls you specify a space-separated string, as described below. For example, to display the Links palette you would use the following command:

```
F_ApiCallClient("WebDAV", "Command OpenLinksPalette");
```

Following are the commands to display WebDAV menus:

F_ApiCallClient() call to perform file actions

`F_ApiCallClient("WebDAV", "Command` *menu_command*`");`
where *menu_command* is one of:

`CancelAllLinkCheckOut`	`OpenServersDialog`
`CancelCheckOut`	`PutFileOnServer`
`CancelCheckOutAll`	`Revert`
`CancelLinkCheckOut`	`RevertAll`
`CancelSelectedLinkCheckOut`	`RevertLink`
`CheckIn`	`Save`
`CheckInAll`	`SaveAll`
`CheckInAllLinks`	`SaveAllLinks`
`CheckInLink`	`SaveAllLinksAs`
`CheckInOutSelected`	`SaveAs`
`CheckInSelectedLinks`	`SaveLink`
`CheckOut`	`SaveLinkAs`
`CheckOutAll`	`SaveSelectedLinks`
`CheckOutAllLinks`	`SaveSelectedLinksAs`
`CheckOutLink`	`Update`
`CheckOutSelectedLinks`	`UpdateAll`
`GetFileFromServer`	`UpdateAllLinks`
`Open`	`UpdateLink`
`OpenLinksPalette`	`UpdateSelectedLinks`
`OpenOptionsDialog`	`UpDownloadSelected`

Call to work with book error logs

When FrameMaker updates a book, if there are error conditions it posts the errors to a log file, and then displays the log. All log management is handled via the BookErrorLog client that is included with the FrameMaker product. This section shows how to post errors via `F_ApiCallClient()`. For more information about posting error messages and displaying the error log, see ‘‘Using the book error log’’ in the *FDK Programmer’s Guide*.

To post a message in the book error log, use the following `F_ApiCallClient()` call:

F_ApiCallClient() calls to post message in book error log

`F_ApiCallClient("BookErrorLog",`

"log -b=[bookId] -d=[docId] -o=[objId] --[text]");

where:

log identifies this as a log message.

- *-b* either the book ID or a document ID; typically the active book. The call creates a unique log for each *-b* ID you pass; if you pass 0 you can create a log that is not associated with any book.
- *-d* either a document ID or an object ID; typically a document associated with a book component. For document ID's the call indents continuous errors for the same document ID underneath the document's filename. This continues until you pass a different document ID. If you pass `0` for the document ID, the call will not indent the errors.
- *-o* is an object in the document represented by the *-d* argument. If you pass both a document ID and an object ID, the call adds a hypertext link, from the error message to the object you specified.
- `--` is the text of the message to appear in the log. To post a time stamp in the message, pass the `FM_PRINT_DATESTAMP` token as the message string.

For example, following code shows how to pass a log entry via `F_ApiCallClient()`; it assumes valid values for `bookId`, `docId`, and `objId`:

```
. . .
StringT log_msg;
log_msg = F_StrNew((UIntT)256);
F_StrTrunc(log_msg, (UIntT)0);
. . .
F_Sprintf(log_msg, "log -b=%d -d=%d -o=%d --%s",
bookId, docId, objId, (StringT)"An error ocurred.");
F_ApiCallClient("BookErrorLog", log_msg);
F_ApiDeallocateString(&log_msg);
. . .
```

CMS Connector FrameWork

8

This chapter describes the structures and enum constants used in the CMS APIs.

CMS API Data Structures and Enum Constants

F_CMSResultT

F_CMSResultT structure specifies the state of a Command's result for API F_ApiCMSCommand

.typedef struct F_CMSResultT

{

```
StatusT status; /* Command's status */

UIntT  opResult;/* Operation's result. If CMS Command needs
CMSTree update, assign F_CMSOpResultT values (See following
enum),else can return any value depending on operation. e.g.
opResult = True/False for FA_CMSIsValidCommand , opResult =
CMSPropertyNewMaxOpCode for FA_CMSGetPropertyMaxOpCode. etc. */

StringT message; /* If operation fails, user can send error
message to FrameMaker.For FA_CMSObjectOpenReadOnly,
FA_CMSObjectEdit command, user can return file-name which is
downloaded. */

F_ObjHandlesT cmsItems;/* List of CMS object */

} F_CMSResultT;
```

F_CMSOpResultT

If F_ApiCMSCommand needs to update the CMS tree, use these enum constant values as part of F_CMSResultT's opResult value.

The possible values of the `F_CMSResultT.opResult` field are:

opResult constant	Meaning
`FV_CMSOpNone`	None
`FV_CMSOpItemUpdated`	CMS item is updated
`FV_CMSOpDependentsUpdated`	Depdendents are updated
`FV_CMSOpDependentsDeleted`	Depdendents are deleted
`FV_CMSOpItemDeleted`	CMS item is deleted
`FV_CMSOpChildAdded`	Child is sdded
`FV_CMSOpRootUpdated`	Root is updated

F_CMSPropertyT

The `F_CMSPropertyT` structure describes a single CMS object property definition

```
typedef struct F_CMSPropertyT
{
UIntT     prop; /* Property Id. Use the value of enum
F_CMSItemPropertyT (See following enum) or custom property added
by the user */
StringT label;/* Label of the property */
BoolT     isMultiValue; /* True if the property is multivalue.
                        */
BoolT     isEditable; /* True if the property is editable. */
F_TypedValsT *values; /* Value of the property */
} F_CMSPropertyT;
```

F_CMSItemPropertyT

Enum constant values exposed for a CMS Object as Properties.

The `F_CMSItemPropertyT` enum specifies the following item properties:

Property	Meaning
`FP_CMSItemProperty_ItemName`	Name of the CMS item

`FP_CMSItemProperty_ItemServerPath`	Server path of the CMS item
`FP_CMSItemProperty_ItemLocalPath`	Local path of the CMS item
`FP_CMSItemProperty_ItemIsCheckedOut`	`True` if CMS item is checked out
`FP_CMSItemProperty_ItemCheckedOutByCurrentUser`	`True` if CMS item is checked out by current user
`FP_CMSItemProperty_ItemIsContainer`	`True` if CMS item is a container

`FP_CMSItemProperty_ItemType`	Type of the CMS item:
	`FV_CMSItemTypeValue_Root`
	`FV_CMSItemTypeValue_Folder`
	`FV_CMSItemTypeValue_File`
	`FV_CMSItemTypeValue_General`

`FP_CMSItemProperty_ItemFileType`	File Type of the CMS item:
	`FV_CMSItemFileTypeValue_Xml`
	`FV_CMSItemFileTypeValue_FmDoc`
	`FV_CMSItemFileTypeValue_Mif`
	`FV_CMSItemFileTypeValue_DitaMap`
	`FV_CMSItemFileTypeValue_FmBook`
	`FV_CMSItemFileTypeValue_Text`
	`FV_CMSItemFileTypeValue_Img`
	`FV_CMSItemFileTypeValue_General`

`FP_CMSItemProperty_ItemVersion`	Version of the CMS item

F_CMSItemTypeValueT

Enum constants used to determine type of CMS Object.

The possible values of the `FP_CMSItemProperty_ItemType` fields are:

Item Type Constant	Meaning
`FV_CMSItemTypeValue_Root`	Item type is Root

`FV_CMSItemTypeValue_Folder`	Item type is Folder
`FV_CMSItemTypeValue_File`	Item type is File
`FV_CMSItemTypeValue_General`	Item type is General

F_CMSItemFileTypeValueT

Enum constants used to determine File-Type of a CMS Object.

The posssible values of the `FP_CMSItemProperty_ItemFileType` fields are:

File Type constant	Meaning
`FV_CMSItemFileTypeValue_Xml`	File type is XML
`FV_CMSItemFileTypeValue_FmDoc`	File type is FM
`FV_CMSItemFileTypeValue_Mif`	File type is MIF
`FV_CMSItemFileTypeValue_DitaMap`	File type is DitaMap
`FV_CMSItemFileTypeValue_DitaTopic`	File type is DitaTopic
`FV_CMSItemFileTypeValue_FmBook`	File type is Book
`FV_CMSItemFileTypeValue_Text`	File type is Txt
`FV_CMSItemFileTypeValue_Img`	File type is Image
`FV_CMSItemFileTypeValue_General`	File type is General

F_CMSPropertiesT

An `F_CMSPropertiesT` structure specifies a set of CMS object propeties.

```
typedef struct F_CMSPropertiesT
{
   IntT   len; /* Number of F_CMSPropertyT */
   F_CMSPropertyT *val; /* Array of F_CMSPropertyT */
} F_CMSPropertiesT;
```

F_CMSMenuItemT

The `F_CMSMenuItemT` structure describes a custom menu definition.
This structure is used for creating a custom menu entry in the context menu available in CMS tree and CMS dialogs.

```
typedef struct F_CMSMenuItemT

{

    IntT  id; /* ID of the Menu item */

    StringT name; /* Name of the Menu item */

    UIntT flags; /* The Menu Type. Use values of enum
F_CMSMenuTypeT (See following enum) */

}F_CMSMenuItemT;
```

F_CMSMenuTypeT

Enum constants used to determine type of a custom menu entry in context menu of a CMS Object.

The user can specify one or more of the following flag constants (using the `OR` expression for multiple flags) into the `F_CMSMenuTypeT.flags` field:

flags constant	Meaning
`FV_CMSMenu_Is_Item`	Custom menu is single item
`FV_CMSMenu_Is_Disabled`	Custom menu is disabled
`FV_CMSMenu_Is_Separator`	Custom menu is separator
`FV_CMSMenu_Is_SubMenu`	Custom menu is of type submenu

F_CMSCheckinParamT

The `F_CMSCheckinParamT` structure describes the checkin parameter.

This strcuture is returned by F_ApiCMSShowCheckinUI API for getting all the user interface state after user accepts the dialog changes by pressing OK button.

```
typedef struct F_CMSCheckinParamT
{
    UIntT    version; /* Use values of enum
                         F_CMSVersioningStrategyT ( See
                         following enum ) */
    StringT  versionLabel;
    StringT  description;
    StringT  checkinComment;
    BoolT    makeThisCurrentVersion;
} F_CMSCheckinParamT;
```

F_CMSVersioningStrategyT

Enum constants used to determine type of versioning of a CMS Object.

The possible values of `F_CMSCheckinParamT.version` field are:

flags constant	Means
`FV_CMSSameVersion`	Same version
`FV_CMSMinorVersion`	Minor version
`FV_CMSMajorVersion`	Major version

F_CMSDeleteParamT

The `F_CMSDeleteParamT` structure describes the delete parameter.

This strcuture is returned by F_ApiCMSShowDeleteUI API for getting all the user interface state after the user accepts the dialog changes by clicking OK.

```
typedef struct F_CMSDeleteParamT
{
    BoolT deleteAllVersion; /* True if user want to delete all
                            the version of a file.*/
    BoolT deleteAllDependents; /* True if user want to delete
                            all the dependents of a file.*/
} F_CMSDeleteParamT;
```

F_CMSInfoT

The `F_CMSInfoT` structure describes a single CMS registration information definition

```
typedef struct F_CMSInfoT
{
   StringT cmsName; /* Name of the CMS */
    F_StringsT userFields; /* List of optional user fields. if
no user fields is specified then its value is NULL
    BoolT userLoginUi; /* False if default Connection manager
                        dialog is used for login*/
} F_CMSInfoT;
```

F_CMSInfosT

An F_CMSInfosT structure specifies a set of CMS registration information.

```
typedef struct F_CMSInfosT
{
   IntT len; /* Number of F_CMSInfoT */
   F_CMSInfoT *val; /* Array of F_CMSInfoT */
} F_CMSInfosT;
```

Error Codes

If the CMS API encounters an error condition, the API assigns an error code to the global variable `FA_errno`. The following table lists the error codes and their meanings. Error codes are also listed in the `fcmsapi.h` header file provided with the FDK .

Error code	Meaning
FE_CMSNameAlreadyRegistered	The API attempted to register a CMS that is already registered.
FE_CMSBadSessionId	The client specified an invalid session ID.
FE_CMSBadObjectId	The client specified an invalid CMS object ID.
FE_CMSSessionFailed	The client failed to create a session.
FE_CMSBadCommandId	The client specified an invalid command ID.
FE_CMSObjectCreationFailed	The F_ApiCMSCreateObject API fails to create a CMS object.
FE_CMSRootObjectExists	The API tried to set a root that already exists.
FE_CMSBadItemFileType	The file type expected by the CMS object does not match the valid file type.
FE_CMSBadItemType	The item type expected by the CMS object does not match the valid item type
FE_CMSBadItemContainerType	The container value expected by the cms object is not properly set
FE_CMSSessionCreationFailed	If Session creation is fails, set status to this value.
FE_CMSIsValidCMSCommand	If user wants FrameMaker to take care of IsValidCMSCommand, set opResult to this value.
FE_CMSFailedLogin	The F_ApiCMSLogin API fails to log into a CMS.
FE_CMSFailedLogout	The F_ApiCMSLogout API fails to log out from a CMS.
FE_CMSFailedCheckout	The F_ApiCMSCheckout API failed to checkout a file.
FE_CMSFailedCheckin	The F_ApiCMSCheckin API failed to check in a file
FE_CMSFailedCancelCheckout	The F_ApiCMSCancelCheckout API fails to cancel checkout of a file.

Error code	Meaning
FE_CMSFailedDelete	The F_ApiCMSDelete API failed to delete a cms object
FE_CMSFailedOpenFile	The F_ApiCMSOpenFile API failed to open a file
FE_CMSFailedUploadObject	The F_ApiCMSUploadObject API failed to upload a file or folder.
FE_CMSFailedDownloadObject	The F_ApiCMSDownloadObject API failed to download a file.
FE_CMSFailedGetItemFrompath	The F_ApiGetCMSObjectFromPath API failed to return a CMS object from server path.

CMS API Functions

F_ApiAllocateCMSInfos

`F_ApiAllocateCMSInfos` allocates memory for an F_CMSInfosT structure and the array of CMS info that it references

Synopsis

```
#include fcmsapi.h
. . .
F_CMSInfosT F_ApiAllocateCMSInfos (IntT numCMSInfos);
```

Arguments

`numCMSInfos` The number of CMS info to allocate

Returns

An `F_CMSInfosT` data structure.
The returned `F_CMSInfosT` structure references memory that is allocated by the API.
Use `F_ApiDeallocateCMSInfos()` to free this memory when you are done with it.
If `F_ApiAllocateCMSInfos()` fails, the API sets the len field of the returned structure to 0.

Example

Example

The following code allocates a memory for a list of 5 CMS info:

```
. . .
F_CMSInfosT cmsinfo;
cmsinfo = F_ApiAllocateCMSInfos(5);
```

F_ ApiDeallocateCMSInfos

Deallocates memory referenced by F_CMSInfosT structure.

Synopsis

```
#include fcmsapi.h
. . .
VoidT F_ApiDeallocateCMSInfos (F_CMSInfosT *cmsInfop)
```

Arguments

`cmsInfop`	The `F_CMSInfosT` structure referencing memory that needs to be deallocated

Returns

`VoidT`

Example

The following code deallocates a CMS info list:

```
. . .
F_CMSInfosT CMSInfo;
. . .
F_ApiDeallocateCMSInfos(&CMSInfo);
```

F_ ApiAllocateCMSProperties

`F_ApiAllocateCMSProperties` allocates memory for an `F_CMSPropertiesT` structure and the array of CMS properties that it references

Synopsis

```
#include fcmsapi.h
. . .
F_CMSPropertiesT F_ApiAllocateCMSProperties (IntT
numCMSProperties);
```

Arguments

`numCMSProperties`	The number of properties in the CMS properties list

Returns

An `F_CMSPropertiesT` data structure.
The returned `F_CMSPropertiesT` structure references memory that is allocated by the API.
Use `F_ApiDeallocateCMSProperties()` to free this memory when you are done with it.
If `F_ApiAllocateCMSProperties()` fails, the API sets the len field of the returned structure to 0.

Example

The following code allocates memory for 5 CMS property list:

```
. . .
F_CMSPropertiesT cmsProp;
cmsProp = F_ApiAllocateCMSProperties(5);
```

F_ApiDeallocateCMSProperties

Deallocates memory referenced by `F_CMSPropertiesT` structure

Synopsis

```
#include fcmsapi.h
. . .
VoidT  F_ApiDeallocateCMSProperties (F_CMSPropertiesT
*cmsPropertiesp);
```

Arguments

`cmsInfop`	The `F_CMSPropertiesT` structure referencing memory that needs to be deallocated

Returns

`VoidT`

Example

The following code deallocates a CMS properties list:

```
. . .
F_CMSPropertiesT cmsProp;
. . .
F_ApiDeallocateCMSProperties(&cmsProp);
```

F_ApiCMSCommand

F_ApiCMSCommand() is a callback that you must define to respond to the user choosing a CMS menu command added by your client. This callback is used for handling various CMS command including user custom commands. User gets multiple objects: F_ObjHandlesT *objectIds

Synopsis

```
#include fcmsapi.h
. . .
VoidT  F_ApiCMSCommand (F_ObjHandleT cmsSessionId,
                        const F_ObjHandlesT *objectIds,
                        IntT command,
                        const F_IdValuePairsT *arguments,
                        F_CMSResultT *statusp);
```

Arguments

`cmsSessionId`	The ID of the CMS session
`cmsObjectId`	The ID of the CMS object
`command`	The CMS command parameter from the `F_CMSCommandsT` enum or `F_ApiCMSAddMenuEntry()` call that creates the custom menu that the use choose
`arguments`	Id value pairs to specify the command arguments. Id takes values from `F_CMSCommandArgsIdT` enum
`statusp`	`F_CMSResultT` structure specifies the state of a command's result. This is also used to update the tree browser accourding to the command

F_CMSCommandsT

Various Command enum constants available for CMS. All commands are valid for F_ApiCMSCommand callback.

The `F_CMSCommandsT` enum specifies the following commands:

Command constants	Meaning
`FA_CMSCreateConnection`	Creates a new CMS connection.This command is triggered when the user clicks Connect in the Connection Manager UI after adding all the credentials.
`FA_CMSSetRootObject`	Sets a root object of a CMS tree. After successful creation of a session, root object is queried using this command.
`FA_CMSCreateConnMgrUI`	Creates a user login UI for a particular cmsId. This command will be triggered only if the user has used F_ApiCMSConfigLoginUI with userLoginUI flag as True).
`FA_CMSGetItemFromPath`	Validates whether an item is valid or not in a server path. This command is triggered if the user has deleted the file from Show common list UI.
`FA_CMSCloseConnection`	Command to close a CMS connection. This command is triggered when user clicks Close connection in the repository browser.
`FA_CMSGetCommandMaxOpCode`	Adds a custom command

FA_CMSGetPropertyMaxOpCode	Adds a custom property
FA_CMSObjectCheckout	Implements checkout functionality.This command is triggered when user clicks checkout
FA_CMSObjectCheckin	Implements Checkin functionality. This command is triggered when user clicks checkin.
FA_CMSObjectCancelCheckout	Implements CancelCheckout functionality. This command is triggered when user clicks CancelCheckout.
FA_CMSObjectEdit	Implements Edit functionality. This command is triggered when user clicks Edit and checkout.-
FA_CMSObjectOpenReadOnly	Implements OpenReadOnly functionality. This command is triggered when user clicks OpenReadOnly.
FA_CMSObjectDelete	Implements Delete functionality. This command is triggered when user clicks Delete.
FA_CMSObjectShowVersion	Implements the ShowVersion functionality. This command is triggered when user clicks ShowVersion.
FA_CMSObjectShowDependents	Implements ShowDependents functionality. This command is triggered when user clicks ShowDependents.
FA_CMSObjectShowProperties	Implements ShowProperties functionality. This command is triggered when user clicks ShowProperties.
FA_CMSObjectShowCheckOutFiles	Implements the ShowCheckOutFiles functionality. This command is triggered when user clicks ShowCheckOutFiles.
FA_CMSObjectDownload	Implements the Download functionality. User gets this command from the Import dialogs.
FA_CMSObjectDownloadItem	Implements the Download functionality. User gets this command from the Open dialog if the file is checked out by the current user.
FA_CMSObjectDownloadForOpen	Implements the Download functionality. User gets this command from Open dialog if file is checked out by other user or not in checked out state.
FA_CMSObjectUploadFile	Implements the Upload File functionality. This command is triggered when user clicks Upload Document.

FA_CMSObjectUploadFolder	Implements the Upload Folder functionality. This command is triggered when user clicks Upload Folder.
FA_CMSObjectGetChildren	Implements the Get Children functionality. Children can be added to a CMS tree using this command. This command is triggered after FA_CMSSetRootObject command.
FA_CMSObjectIsSame	Command to validate that the two CMS Objects are same.
FA_CMSObjectRefresh	Implements the Refresh functionality. This command is triggered when user clicks Refresh.
FA_CMSSimpleSearch	Implements the Basic Search functionality. This command is triggered when user clicks Search Repository.
FA_CMSAdvancedSearch	Implements the Advance Search functionality. This command is triggered when user clicks Advance Search.
FA_CMSGetItems	Implements the Get item functionality. CMS item to a show common list UI can be added using this command. This is triggered from show common list UI.
FA_CMSBuildContextMenu	Implements the Build context menu. When user right clicks on tree browser of a node, this command is triggered. This command can be used to Enable/Disable command for a particular object Id.
FA_CMSIsValidCommand	Command to validate a CMS command. This command also contains the context as argument string. If Menu is called for browser, context is “”. Whereas for dialog, context is the name of dialog, such as “Show Version.”

F_CMSCommandArgsIdT

Enum constants used to verify various arguments of F_ApiCMSCommand callback.

The F_CMSCommandArgsIdT enum specifies the following command arguments

Value for flags	Meaning
FV_CMSCommandNameId	F_ApiCMSLogin() uses this value to set “Name of the CMS connection”

`FV_CMSCommandConnTypeId`	`F_ApiCMSLogin()` uses this value to set "Connection Type"
`FV_CMSCommandServerId`	`F_ApiCMSLogin()` uses this value to set "Name of the Server"
`FV_CMSCommandUserNameId`	`F_ApiCMSLogin()` uses this value to set "Name of the User"
`FV_CMSCommandPasswordId`	`F_ApiCMSLogin()` uses this value to set "Password"
`FV_CMSCommandUserField1`	`F_ApiCMSLogin()` uses this value to set "User Field1"
`FV_CMSCommandUserField2`	`F_ApiCMSLogin()` uses this value to set "User Field2"
`FV_CMSCommandFilePathId`	Argument for File Path to upload
`FV_CMSCommandSearchStringId`	Argument for Basic Search String
`FV_CMSCommandAdvancedSearchStringId`	Argument for Advance Search String
`FV_CMSCommandId`	Argument for Command ID
`FV_CMSContextMenuId`	Argument for Context Menu ID
`FV_CMSContextMenuString`	Argument for Context Menu String
`FV_CMSSilentOperation`	CMS automation API's uses this value to set "Silent operation" parameter
`FV_CMSCommandCheckoutWithDescendentId`	`F_ApiCMSCheckout()` uses this value to set "Checkout with descendent" parameter
`FV_CMSCommandCheckinMakeCurrentVersionId`	`F_ApiCMSCheckin()` uses this value to set "Make Current Version" parameter
`FV_CMSCommandCheckinKeepLocalCopyId`	`F_ApiCMSCheckin()` uses this value to set "Keep Local Copy" parameter
`FV_CMSCommandCheckinMinorVersionId`	`F_ApiCMSCheckin()` uses this value to set "Minor Version" parameter
`FV_CMSCommandCheckinVersionLabelId`	`F_ApiCMSCheckin()` uses this value to set "Version Label" parameter
`FV_CMSCommandCheckinDescriptionId`	`F_ApiCMSCheckin()` uses this value to set "Description" parameter
`FV_CMSCommandCheckinCommentId`	`F_ApiCMSCheckin()` uses this value to set "Checkin Comment" parameter

FV_CMSCommandDeleteAllVersionId	F_ApiCMSDelete() uses this value to set "Delete All Version" parameter
FV_CMSCommandDeleteAllDependentsId	F_ApiCMSDelete() uses this value to set "Delete All Dependents" parameter
FV_CMSCommandOpenReadOnlyId	F_ApiCMSOpenFile() uses this value to set "Oper Read Only" parameter
FV_CMSCommandSilentOpenId	F_ApiCMSOpenFile() uses this value to set "Silent Open" parameter

Returns

VoidT

If F_ApiCMSCommand() fails, the API assigns following values to FA_errno:

FA_errno value	Meaning
FE_CMSBadSessionId	The client specified an invalid session ID.
FE_CMSBadObjectId	The client specified an invalid CMS object ID.
FE_CMSBadCommandId	The client specified an invalid command ID.
FE_BadParameter	The function call specified an invalid parameter.

Responding to the user choosing a command

Whenever the user chooses a CMS menu item for a CMS command created by your client, FrameMaker attempts to call your client's F_ApiCMSCommand() function.

Example

The following code provides an F_ApiCMSCommand() function to respond to the user choosing the CMS commands:

```
. . .
VoidT F_ApiCMSCommand(F_ObjHandleT cmsSessionId,const
F_ObjHandlesT *objectIds, IntT command, const F_IdValuePairsT
*arguments, F_CMSResultT *statusp)
```

```
switch(command)
{
case FA_CMSObjectOpenReadOnly: /* Code to open a file in read only
mode goes here. */
break;
case FA_CMSObjectDelete: /* Code to delete a CMS object goes here.
*/
break;
}
}
```

F_ApiCMSRegister

Registers a CMS client.

Call this API within the F_ApiInitialize() section of the FDK client. After the client is registered in the [API Clients] section of the maker.ini file, the name of the CMS is shown in “Choose Connection” drop-down of the Connection Manager dialog.

Synopsis

```
#include fcmsapi.h
. . .
F_ObjHandleT F_ApiCMSRegister (ConStringT cmsName);
```

Arguments

`cmsName`	Name of the CMS to register

Returns

Returns the ID of the new CMS if it succeeds, or an error code if an error occurs.

If `F_ApiCMSRegister()` fails, the API assigns following values to `FA_errno`

FA_errno value	Meaning
`FE_CMSNameAlreadyRegistered`	The API attempt to register a CMS that is already registered.
`FE_BadParameter`	The function call specified an invalid parameter.

Example

The following code register a CMS “Adobe CQ”

```
. . .
VoidT F_ApiInitialize(IntT initialization)
{
F_ObjHandleT cmsId;
. . .

cmsId = F_ApiCMSRegister((ConStringT)"Adobe CQ");
}
```

F_ApiCMSCreateObject

Creates a CMS object.

Synopsis

```
#include fcmsapi.h
. . .
F_ObjHandleT F_ApiCMSCreateObject (F_ObjHandleT cmsSessionId);
```

Arguments

`cmsSessionId` The ID of the CMS session

Returns

Returns the ID of the new CMS object if it succeeds, or an error code if an error occurs.

If `F_ApiCMSCreateObject()` fails, the API assigns following values to `FA_errno`

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID.
`FE_CMSObjectCreationFailed`	API failed to create a cms object.

Example

The following code creates a CMS object:

```
. . .
F_ObjHandlesT *objHandles = (F_ObjHandlesT
*)F_Alloc(sizeof(F_ObjHandlesT),DSE);
objHandles->val[0] = F_ApiCMSCreateObject(cmsSessionId);
. . .
```

F_ApiCMSEnableCommand

Enables the specified CMS command in the context menu of the CMS tree within FrameMaker.

Synopsis

```
#include fcmsapi.h
. . .
VoidT  F_ApiCMSEnableCommand (F_ObjHandleT cmsSessionId,
                              F_ObjHandleT objectId,
                              UIntT commandId);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`commandId`	The command to enable

Returns

VoidT.

If `F_ApiCMSEnableCommand()` fails, the API assigns following values to `FA_errno`

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID.
`FE_CMSBadObjectId`	The client specified an invalid cms object ID.
`FE_CMSBadCommandId`	The client specified an invalid command ID.

Example

The following code enables Edit , Delete and Checkin command:

```
. . .
FA_errno=0;
F_ApiCMSEnableCommand(cmsSessionId,objectId,FA_CMSObjectEdit);
F_ApiCMSEnableCommand(cmsSessionId,objectId,FA_CMSObjectDelete);
F_ApiCMSEnableCommand(cmsSessionId,objectId,FA_CMSObjectCheckin)
;
```

F_ApiCMSDisableCommand

Disables the specified CMS command in the context menu of the CMS tree within FrameMaker.

Synopsis

```
#include fcmsapi.h
. . .
VoidT  F_ApiCMSDisableCommand (F_ObjHandleT cmsSessionId,
                               F_ObjHandleT objectId,
                               UIntT commandId);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`commandId`	The command to disable

Returns

VoidT.

If `F_ApiCMSDisableCommand()` fails, the API assigns following values to `FA_errno`

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID.
`FE_CMSBadObjectId`	The client specified an invalid cms object ID.
`FE_CMSBadCommandId`	The client specified an invalid command ID.

Example

The following code disables OpenReadOnly , Delete and Checkout command:

```
. . .
FA_errno=0;
F_ApiCMSDisableCommand(cmsSessionId,objectId,FA_CMSObjectOpenRea
dOnly);
F_ApiCMSDisableCommand(cmsSessionId,objectId,FA_CMSObjectDelete)
;
F_ApiCMSDisableCommand(cmsSessionId,objectId,FA_CMSObjectCheckou
t);
```

F_ApiCMSAddMenuEntry

Adds a custom menu entry in the context menu within the FrameMaker interface.

Synopsis

```
#include fcmsapi.h
. . .
F_ObjHandleT F_ApiCMSAddMenuEntry (F_ObjHandleT menuId,
                            const F_CMSMenuItemT *menuEntryp);
```

Arguments

`menuId`	The ID of the Parent menu
`menuEntryp`	The `F_CMSMenuItemT` structure describes a custom menu definition

Returns

Returns the ID of the newly created custom menu entry if it succeeds, or an error code if an error occurs.

If `F_ApiCMSAddMenuEntry()` fails, the API assigns following values to `FA_errno`:

FA_errno value	Meaning
`FE_CMSBadObjectId`	The client specified an invalid menu ID.
`FE_BadParameter`	The function call specified an invalid parameter.

Example

The following code creates a custom menu “New Folder” inside a parent menu referred by menuId.

```
. . .
F_CMSMenuItemT *menu = (F_CMSMenuItemT
*)F_Alloc(sizeof(F_CMSMenuItemT),DSE);
menu->id = FA_CMSCommandMax+1;;
menu->name = (StringT)"New Folder";
menu->flags = FV_CMSMenu_Is_Item;
F_ApiCMSAddMenuEntry(menuId,menu);
```

F_ApiCMSGetProperties

Gets the properties of a CMS object.

Synopsis

```
#include fcmsapi.h
. . .
F_PropValsT F_ApiCMSGetProperties (F_ObjHandleT cmsSessionId,
                                   F_ObjHandleT objectId);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`objectId`	The ID of the CMS Object

Returns

Returns the property list (an `F_PropValsT` data structure) with the CMS properties if it succeeds or an error code if an error occurs

If `F_ApiCMSGetProperties()` fails, the API assigns following values to `FA_errno`:

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID.
`FE_CMSBadObjectId`	The client specified an invalid cms object ID.

Example

The following code gets the properties of a CMS object:

```
. . .
IntT counter;
F_PropValsT propVals;
FA_errno = 0;
propVals=F_ApiCMSGetProperties(cmsSessionId,objectId);
F_Printf ((ChannelT )NULL,"Total properties=%d",propVals.len);
for (counter=0;counter<(IntT)propVals.len;counter++)
{
 if (FT_String==propVals.val[counter].propVal.valType)
  F_Printf ( (ChannelT )NULL,"[%s]=%s\n",propVals.val[counter].
propIdent.name,propVals.val[counter].propVal.u.sval);
  else if (FT_Integer==propVals.val[counter].propVal.valType)
  F_Printf ( (ChannelT )NULL, "[%s]=%d\n",propVals.val[counter].
propIdent.name,propVals.val[counter].propVal.u.ival);
}
```

F_ApiCMSGetProperty

Gets a specified property of a CMS object.

Synopsis

```
#include fcmsapi.h
. . .
F_PropValT F_ApiCMSGetProperty (F_ObjHandleT cmsSessionId,
                                F_ObjHandleT objectId,
                                const F_PropIdentT *propertyId);
```

Arguments

`cmsSessionId`	The ID of the CMS Session.
`objectId`	The ID of the CMS Object.
`propertyId`	F_PropIdentT which allows user to specify property identifier as Integer value OR string value based on the CMS. For e.g. Documentum works on object name, whereas Generic CMS works on Integer based identifier ID of the CMS Object.

Returns

Returns the specified property of a CMS object as a `F_PropValT` data structure if it succeeds, or an error code if an error occurs.

If `F_ApiCMSGetProperty()` fails, the API assigns following values to `FA_errno:`

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID.
`FE_CMSBadObjectId`	The client specified an invalid cms object ID.
`FE_BadParameter`	The function call specified an invalid parameter.

Example

The following code gets the name of a CMS object:

```
. . .
FA_errno = 0;
F_PropIdentT  propertyid;
propertyid.num=FP_CMSItemProperty_ItemName;
F_PropValT ItemName=F_ApiCMSGetProperty( cmsSessionId, objectId,
&propertyid);
F_Printf((ChannelT )NULL,"%s\n",ItemName.propVal.u.sval);
```

F_ApiCMSSetProperties

Sets multiple properties of a CMS object

Synopsis

```
#include fcmsapi.h
. . .
VoidT  F_ApiCMSSetProperties (F_ObjHandleT cmsSessionId,
                              F_ObjHandleT objectId,
                              const F_PropValsT *propVals)
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`propVals`	A property list that specifies the CMS properties . Properties are added as propvals with the identifier as F_CMSItemPropertyT enum and value.

Returns

VoidT

If `F_ApiCMSSetProperties()` fails, the API assigns following values to `FA_errno`:

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID.
`FE_CMSBadObjectId`	The client specified an invalid cms object ID.
`FE_CMSRootObjectExists`	The API tries to set a root which already exists.
`FE_CMSBadItemFileType`	The file type expected by the cms object does not match the valid file type.
`FE_CMSBadItemType`	The item type expected by the cms object does not match the valid item type
`FE_CMSBadItemContainer Type`	The container value expected by the CMS object is not properly set

Valid Scenarios for object properties

Following are the valid API scenarios of properties on a particular object.

ItemType	ItemFileType	IsContainer
`Root`	`General`	`True`
`Folder`	`General`	True
`File`	Any Value	False
`General`	Any Value	Any Value

Example
The following code sets the properties of a CMS object:

```
. . .
F_PropValsT propVals;
F_ObjHandlesT *objHandles = (F_ObjHandlesT
*)F_Alloc(sizeof(F_ObjHandlesT),DSE);
objHandles->len = 1;
objHandles->val = (F_ObjHandleT *)F_Alloc(objHandles-
>len*sizeof(F_ObjHandleT),DSE);
objHandles->val[0] = F_ApiCMSCreateObject(cmsSessionId);

propVals = F_ApiAllocatePropVals(4);
if(propVals.len != 0)
{
propVals.val[0].propIdent.num = FP_CMSItemProperty_ItemName;
propVals.val[0].propIdent.name = F_StrCopyString("Name");
propVals.val[0].propVal.valType = FT_String;
propVals.val[0].propVal.u.sval =  F_StrCopyString("root");

propVals.val[1].propIdent.num=FP_CMSItemProperty_ItemServerPath;
propVals.val[1].propIdent.name = F_StrCopyString("Server Path");
propVals.val[1].propVal.valType = FT_String;
propVals.val[1].propVal.u.sval =  F_StrCopyString("/");
propVals.val[2].propIdent.num=FP_CMSItemProperty_ItemIsContainer
;
propVals.val[2].propIdent.name = F_StrCopyString("IsContainer");
propVals.val[2].propVal.valType = FT_Integer;
propVals.val[2].propVal.u.ival =  True;
propVals.val[3].propIdent.num = FP_CMSItemProperty_ItemType;
propVals.val[3].propIdent.name = F_StrCopyString("Type");
propVals.val[3].propVal.valType = FT_Integer;
propVals.val[3].propVal.u.ival =  FV_CMSItemTypeValue_Root;}
FA_errno = 0;
F_ApiCMSSetProperties(cmsSessionId,objHandles->val[0],
&propVals);
```

F_ApiCMSSetProperty

Sets a single property for a CMS object.

```
. . .
VoidT  F_ApiCMSSetProperty (F_ObjHandleT cmsSessionId,
                            F_ObjHandleT objectId,
                            const F_PropValT *propVal);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`propVal`	The specified property of a CMS object as a `F_PropValT` data structure

Returns

`VoidT`

If `F_ApiCMSSetProperty()` fails, the API assigns following values to `FA_errno`:

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID.
`FE_CMSBadObjectId`	The client specified an invalid cms object ID.
`FE_CMSRootObjectExists`	The API tries to set a root which already exists.
`FE_CMSBadItemFileType`	The file type expected by the cms object does not match the valid file type.
`FE_CMSBadItemType`	The item type expected by the cms object does not match the valid item type
`FE_CMSBadItemContainer Type`	The container value expected by the cms object is not properly set

Example

The following code gets the checkedout property of a CMS object:

```
. . .
F_PropValT *prop;
prop=(F_PropValT *)F_Alloc(sizeof(F_PropValT), NO_DSE);
prop->propIdent.num= FP_CMSItemProperty_ItemIsCheckedOut;
prop->propIdent.name = F_StrCopyString("ItemIsCheckedOut");
prop->propVal.valType = FT_Integer;
prop->propVal.u.ival =  TRUE;
FA_errno=0;
F_ApiCMSSetProperty(cmsSessionId,objectIds->val[0], prop);
```

F_ApiCMSConfigLoginUI

Configures the CMS Connector Manager dialogs within the FrameMaker interface

Synopsis

```
#include fcmsapi.h
. . .
VoidT  F_ApiCMSConfigLoginUI (F_ObjHandleT cmsId,
                              const F_StringsT *userFields,
                              BoolT userLoginUI);
```

Arguments

`cmsId`	The ID of the CMS
`userFields`	Optional user fields with strings. User can add upto two user fields.
`userLoginUI`	True if user want to impement his own custom connection manager dialog

Returns

`VoidT`

If `F_ApiCMSConfigLoginUI()` fails, the API assigns following values to `FA_errno`

FA_errno value	Meaning
`FE_BadParameter`	The function call specified an invalid parameter.

Example

The following code configure the connection manager dialog for “Adobe CQ”

```
. . .
VoidT F_ApiInitialize(IntT initialization)
{
F_StringsT *userFields = (F_StringsT
*)F_Alloc(sizeof(F_StringsT), NO_DSE);
userFields->len = 1;
userFields->val = (StringT *)F_Alloc(sizeof(StringT) *
userFields->len, NO_DSE);
userFields->val[0] =F_ApiCopyString((ConStringT)"WorkSpace:");


F_ObjHandleT cmsId = F_ApiCMSRegister((ConStringT)"Adobe CQ");
F_ApiCMSConfigLoginUI(cmsId,userFields, False);
F_Free(userFields);

}
```

F_ApiCMSShowCheckoutUI

Displays the checkout dialog for a CMS object

Synopsis

```
#include fcmsapi.h
. . .
BoolT  F_ApiCMSShowCheckoutUI(F_ObjHandleT sessionId,
                              F_ObjHandleT objectId,
                              UIntT hideUiItems);
```

Arguments

`sessionId`	The ID of the CMS session
`objectId`	The ID of the CMS Object
`hideUiItems`	Parameter to customize the checkout dialog. It can take the values from F_CMSCustomizeCheckoutUI enum

F_CMSCustomizeCheckoutUI

Enum constants used to customize CMS Object's Checkout user interface.

The possible values of the `hideUiItems` field are:

Value for flags	Meaning
`FV_CMSCheckoutUI_Id_ShowDependents`	Flag to hide “Show dependents” checkbox

Returns

Returns True if “Show dependents” option is checked if it succeeds or an error code if an error occurs

If `F_ApiCMSShowCheckoutUI()` fails, the API assigns following values to `FA_errno`:

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID
`FE_CMSBadObjectId`	The client specified an invalid cms object ID
`FE_BadParameter`	The function call specified an invalid parameter.

Example

The following code displays the checkout dialog with show dependents option:

```
. . .
BoolT Status;

FA_errno=0;

Status=F_ApiCMSShowCheckoutUI(cmsSessionId, objectIds->val[0],
0);
```

F_ApiCMSShowCheckinUI

Displays the Check-in dialog for a CMS object

Synopsis

```
#include fcmsapi.h
. . .
F_CMSCheckinParamT F_ApiCMSShowCheckinUI(F_ObjHandleT sessionId,
                            F_ObjHandleT objectId,
                            UIntT hideUiItems);
```

Arguments

`sessionId`	The ID of the CMS session
`objectId`	The ID of the CMS Object
`hideUiItems`	Parameter to customize the checkin dialog. It can take the values from F_CMSCustomizeCheckinUI enum

F_CMSCustomizeCheckinUI

Enum constants used to customize CMS Object's Checkin user interface.

The user can specify one or more of the following flag constants (using the `OR` expression for multiple flags) into the `hideUiItems` field:

hideUiItems constants	Meaning
`FV_CMSCheckinUI_Id_SameVersion`	Flag to hide “Same version” radio button
`FV_CMSCheckinUI_Id_MinorVersion`	Flag to hide “Minor version” radio button

FV_CMSCheckinUI_Id_MajorVersion	Flag to hide “Major version” radio button
FV_CMSCheckinUI_Id_VersionLabel	Flag to hide “Version Label” text field
FV_CMSCheckinUI_Id_Description	Flag to hide “Description” text field
FV_CMSCheckinUI_Id_CheckinComment	Flag to hide “Checkin comment” text field
FV_CMSCheckinUI_Id_MakeThisCurrentVersion	Flag to hide “Make this current version” checkbox

Returns

Retiurns the F_CMSCheckinParamT structure for further operation in the client if it succeeds or an error code if an error occurs

If F_ApiCMSShowCheckinUI() fails, the API assigns following values to FA_errno:

FA_errno value	Meaning
FE_CMSBadSessionId	The client specified an invalid session ID
FE_CMSBadObjectId	The client specified an invalid cms object ID
FE_BadParameter	The function call specified an invalid parameter.

Example

The following code displays the checkin dialog with all fields：

```
. . .
F_CMSCheckinParamT param;
FA_errno=0;
param=F_ApiCMSShowCheckinUI( cmsSessionId,objectIds->val[0], 0);
```

F_ApiCMSShowCancelCheckoutUI

Displays the Cancel Check out dialog for a CMS object

Synopsis

```
#include fcmsapi.h
. . .
VoidT  F_ApiCMSShowCancelCheckoutUI (F_ObjHandleT
             sessionId,F_ObjHandleT objectId);
```

Arguments

`cmsSessionId`	The ID of the CMS session
`cmsObjectId`	The ID of the CMS object

Returns

`VoidT`

If `F_ApiCMSShowCancelCheckoutUI()` fails, the API assigns following values to `FA_errno`

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID
`FE_CMSBadObjectId`	The client specified an invalid cms object ID

Example
The following code displays the cancel checkout dialog:

```
. . .
F_ApiCMSShowCancelCheckoutUI( cmsSessionId, objectIds->val[0]);
```

F_ApiCMSShowDeleteUI

Displays the Delete dialog for a CMS object

Synopsis

```
#include fcmsapi.h

. . .
F_CMSDeleteParamT F_ApiCMSShowDeleteUI (F_ObjHandleT
                    cmsSessionId,F_ObjHandleT
                    objectId,UIntT hideUiItems);
```

Arguments

`sessionId`	The ID of the CMS session
`objectId`	The ID of the CMS Object
`hideUiItems`	Parameter to customize the delete dialog. It can take the values from F_CMSCustomizeDeleteUI enum

F_CMSCustomizeDeleteUI

Enum constants used to customize CMS Object's Delete user interface.

The user can specify one or more of the following flag constants (using the `OR` expression for multiple flags) into the `hideUiItems` field:

Value for flags	Meaning
FV_CMSDeleteUI_DeleteAllVersion	Flag to hide “Delete all the version of a file” checkbox
FV_CMSDeleteUI_DeleteAllDependents	Flag to hide “Delete all the dependents of a file” checkbox

Returns

Retiurns the `F_CMSDeleteParamT` structure for further operation in the client if it succeeds or an error code if an error occurs

If `F_ApiCMSShowDeleteUI()` fails, the API assigns following values to `FA_errno:`

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID
`FE_CMSBadObjectId`	The client specified an invalid cms object ID
`FE_BadParameter`	The function call specified an invalid parameter.

Example
The following code displays the delete dialog with all fields:

```
. . .
F_CMSDeleteParamT param;
FA_errno=0;
param=F_ApiCMSShowDeleteUI( cmsSessionId, objectIds->val[0],0);
```

F_ApiCMSShowCommonListUI

Displays the list-based dialogs such as “Show Version”, “Show Checked out files”, “Show dependents” and “Show result”. These items are fetched using the F_ApiCMSGetItems CMS command.

Synopsis

```
#include fcmsapi.h
. . .
BoolT  F_ApiCMSShowCommonListUI (F_ObjHandleT
                    cmsSessionId,F_ObjHandleT objectId,
                    UIntT commandId,ConStringT title,
                    const F_TypedValsT *columnProperties);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`commandId`	The ID of the CMS command
`title`	The Title of the Show Dialog
`columnProperties`	Properties Columns to show

Returns

Returns `FE_Success` if it succeeds, or an error code if an error occurs.

If `F_ApiCMSShowCommonListUI()` fails, the API assigns following values to `FA_errno`

FA_errno value	**Meaning**
`FE_CMSBadSessionId`	The client specified an invalid session ID.
`FE_CMSBadObjectId`	The client specified an invalid cms object ID.
`FE_CMSBadCommandId`	The client specified an invalid command ID.
`FE_BadParameter`	The function call specified an invalid parameter.

Example

The following code displays the “Show Version” dialog with 2 column “Name” and “Server Path”

```
. . .
F_TypedValsT columnProperties;
F_TypedValT columnProperty[2];
columnProperties.len = 2;
columnProperties.val = (F_TypedValT
*)F_Alloc(columnProperties.len * sizeof(F_TypedValT),DSE);
```

```
for(IntT i=0;i<columnProperties.len; i++)
{
columnProperty[i].valType = FT_Vals;
columnProperty[i].u.valsval = (F_TypedValsT *)F_Alloc(1 *
sizeof(F_TypedValsT),DSE);
columnProperty[i].u.valsval->len = 2;
columnProperty[i].u.valsval->val = (F_TypedValT *)F_Alloc(2 *
sizeof(F_TypedValT),DSE);
switch(i)
{
case 0:
columnProperty[i].u.valsval->val[0].valType = FT_Integer;
columnProperty[i].u.valsval->val[0].u.ival =
FP_CMSItemProperty_ItemName;
columnProperty[i].u.valsval->val[1].valType = FT_String;
columnProperty[i].u.valsval->val[1].u.sval = F_StrCopyString(
(ConStringT) "Name");
break;
case 1:
columnProperty[i].u.valsval->val[0].valType = FT_Integer;
columnProperty[i].u.valsval->val[0].u.ival =
FP_CMSItemProperty_ItemServerPath;
columnProperty[i].u.valsval->val[1].valType = FT_String;
columnProperty[i].u.valsval->val[1].u.sval = F_StrCopyString(
(ConStringT) "Server Path");
break;
}
columnProperties.val[i] = columnProperty[i];
}
BoolT status=F_ApiCMSShowCommonListUI(cmsSessionId, objectId,
command, "Show Version", &columnProperties);
```

F_ApiCMSShowPropertyUI

Displays the Property dialog for a CMS object

Synopsis

```
#include fcmsapi.h
. . .
F_CMSPropertiesT F_ApiCMSShowPropertyUI(F_ObjHandleT
                 cmsSessionId,F_ObjHandleT objectId,
                 const F_CMSPropertiesT *props);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`props`	`F_CMSPropertiesT` structure specifies a set of CMS object propeties. Set NULL if user wants to show default properties

Returns

Returns `F_CMSPropertiesT` strucuture contaning a set of CMS object properties if it succeeds, or an error code if an error occurs.

If `F_ApiCMSShowPropertyUI()` fails, the API assigns following values to `FA_errno`

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID
`FE_CMSBadObjectId`	The client specified an invalid cms object ID
`FE_BadParameter`	The function call specified an invalid parameter.

Example

The following code displays the “Show Property” dialog with default properties:

```
. . .
F_CMSPropertiesT retProps;
retProps = F_ApiCMSShowPropertyUI(cmsSessionId, objectIds-
>val[0], NULL);
```

F_ApiCMSShowBrowseRepositoryUI

Displays repository browser dialog based on flag “showContainerOnly”

Synopsis

```
#include fcmsapi.h
. . .
F_ObjHandleT F_ApiCMSShowBrowseRepositoryUI(F_ObjHandleT
cmsSessionId, BoolT showContainerOnly);
```

Arguments

`cmsSessionId`	The ID of the CMS session
`showContainerOnly`	True if only container item is shown False if all items are shown

Returns

```
Returns the ID of the selected CMS object
```

Example

The following code displays the repository browser dialog for only container items:

```
. . .
F_ObjHandleT cmsObj;
FA_errno=0;
cmsObj=F_ApiCMSShowBrowseRepositoryUI(cmsSessionId,(BoolT)1);
if (FA_errno==0)
{
if (cmsObj)
F_ApiAlert("item selected", FF_ALERT_CONTINUE_NOTE);;
else
F_ApiAlert("No item selected", FF_ALERT_CONTINUE_NOTE);
}
else
F_ApiAlert("Canceled button is clicked", FF_ALERT_CONTINUE_NOTE);
```

F_ApiCMSGetCmsIdFromName

Gets the CMS registration id from CMS name.

Synopsis

```
#include fcmsapi.h
. . .
F_ObjHandleT F_ApiCMSGetCmsIdFromName (ConStringT cmsName);
```

Arguments

`cmsName`	The Name of the CMS

Returns

Returns the registration ID of the specified CMS if it succeeds, or an error code if an error occurs.

If `F_ApiCMSGetCmsIdFromName()` fails, the API assigns following values to `FA_errno`:

FA_errno value	Meaning
`FE_BadParameter`	The function call specified an invalid parameter.

Example

The following code gets the CMS registration id of `"Adobe CQ"`

```
. . .
F_ObjHandlesT cmsId;
cmsId = F_ApiCMSGetCmsIdFromName("Adobe CQ");
```

F_ApiCMSGetCMSInfo

Gets the CMS information for a particular CMS registration id.

Synopsis

```
#include fcmsapi.h
.. .
F_CMSInfoT    F_ApiCMSGetCMSInfo(F_ObjHandleT cmsId);
```

Arguments

`cmsId`	The registration ID of the CMS

Returns

Returns the `F_CMSInfoT` structure contaning a single CMS registration information if it succeeds or an error code if an error occurs

If `F_ApiCMSGetCMSInfo()` fails, the API assigns following values to `FA_errno`:

FA_errno value	Meaning
`FE_BadParameter`	The function call specified an invalid parameter.

Example

The following code gets the CMS information for a particular registered CMS:

```
. . .
F_CMSInfoT CMSInfo= F_ApiCMSGetCMSInfo(cmsId);
```

F_ApiCMSGetCmsIdFromSession

Gets the CMS registration id from CMS Session id

Synopsis

```
#include fcmsapi.h
. . .
F_ObjHandleT  F_ApiCMSGetCmsIdFromSession(F_ObjHandleT
cmsSessionId);
```

Returns

Returns the CMS registration ID if it succeeds, or an error code if an error occurs.

If `F_ApiCMSGetCmsIdFromSession()` fails, the API assigns following values to `FA_errno`

FA_errno value	Meaning
`FE_CMSBadSessionId`	The client specified an invalid session ID

Example

The following code gets the CMS registration id from CMS session id

```
. . .
F_ObjHandlesT cmsId;
cmsId = F_ApiCMSGetCmsIdFromSession(cmsSessionId);
```

F_ApiCMSGetCmsInfoList

Gets entire list of CMS information for all registered CMS

Synopsis

```
#include fcmsapi.h
. . .
F_CMSInfosT  F_ApiCMSGetCmsInfoList ();
```

Arguments

None

Returns

Returns the `F_CMSInfosT` structure contaning CMS information for all registered CMS.

Example

The following code gets the entire list of CMS information for all registered CMS:

```
. . .

F_CMSInfosT CMSInfo= F_ApiCMSGetCmsInfoList();
```

APIs to automate the CMS connectors functionality

9

This chapter describes the CMS APIs used automate the CMS operations after a CMS connector is integrated with FrameMaker.

F_ApiCMSLogin

Logs into a particular CMS based on the connection details

Synopsis

```
#include fcmsapi.h
. . .
F_ObjHandleT F_ApiCMSLogin(const F_IdValuePairsT *setVal);
```

Arguments

`setVal`	Id value pairs to specify the connection parameter. The valid Ids are: FV_CMSCommandNameID - Name of the connection FV_CMSCommandConnTypeId -Connection Type FV_CMSCommandServerId -Server Name FV_CMSCommandUserNameId -User Name FV_CMSCommandPasswordId- Password FV_CMSCommandUserField1- Optional User Field1 FV_CMSCommandRepositoryId- Repository name for documentum FV_CMSCommandUserField2-Optional User Field2

Returns

Returns the handle of the new CMS connection if the operation is successful. Else sets `FA_errno` to `FE_CMSFailedLogin`.

Example

The following code logs into the CRX and returns CMS connection id:

```
. . .
#define ASSIGN_ID_STR(COUNT,ID,STR) tval->val[COUNT].id = ID;\
                tval->val[COUNT].value.valType = FT_String; \
ASSIGN_ID_STR(0, FV_CMSCommandConnTypeId,"CRX");
ASSIGN_ID_STR(1, FV_CMSCommandNameId, "CRX");
ASSIGN_ID_STR(2, FV_CMSCommandServerId,
"http://localhost:7402/crx/server");
ASSIGN_ID_STR(3, FV_CMSCommandUserNameId, "admin");
ASSIGN_ID_STR(4, FV_CMSCommandPasswordId, "admin");
ASSIGN_ID_STR(5, FV_CMSCommandUserField1, "crx.default");
ASSIGN_ID_STR(6, FV_CMSCommandUserField2, "crx");
connId = F_ApiCMSLogin(tval);
```

F_ApiCMSLogout

Logs out the user from a particular CMS connection

Synopsis

```
#include fcmsapi.h
. . .
StatusT F_ApiCMSLogout(F_ObjHandleT cmsSessionId);
```

Arguments

`cmsSessionId` The ID of the CMS session

Returns

Returns `FE_Success` if the operation is successful, else sets FA_errno to `FE_CMSFailedLogout`

Example

The following code logs out the CMS connection:

```
. . .
FA_errno=0;
F_ObjHandleT connIdconnId = F_ApiCMSLogin(tval);
StatusT result=F_ApiCMSLogout(connId);
```

F_ApiCMSCheckout

Checks out a file from the CMS

Synopsis

```
#include fcmsapi.h
. . .
StatusT   F_ApiCMSCheckout(F_ObjHandleT cmsSessionId,
          F_ObjHandleT cmsObjectId,BoolT rootWithDescendants);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`rootWithDescendants`	`True` if checked out root with descendants

Returns

Returns `FE_Success` if the operation is successful. Else sets FA_errno to `FE_CMSFailedCheckout`

Example

The following code checkouts a file with its descendants:

```
. . .
FA_errno=0;
F_ObjHandleT objectId = 0;
ConStringT urlPathOrObjId =
F_ApiCopyString((ConStringT)"/libs/source.fm");
F_ObjHandleT cmsObjId = F_ApiGetCMSObjectFromPath(ConnId,
urlPathOrObjId);
BoolT downloadDescendents = True;
if (cmsObjId)
{
StatusT result=F_ApiCMSCheckout(ConnId, cmsObjId,
downloadDescendents);
objectId=cmsObjId;
}
```

F_ApiCMSCheckin

Checks in a file into the CMS

Synopsis

```
#include fcmsapi.h
. . .
StatusT F_ApiCMSCheckin(F_ObjHandleT cmsSessionId,F_ObjHandleT
                objectId,const F_IdValuePairsT *checkinParam);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`checkinParam`	Id value pairs to specify the checkin parameter. The valid Ids are: FV_CMSCommandCheckinMakeCurrentVersionId FV_CMSCommandCheckinKeepLocalCopyId FV_CMSCommandCheckinMinorVersionId FV_CMSCommandCheckinVersionLabelId FV_CMSCommandCheckinDescriptionId FV_CMSCommandCheckinCommentId

Returns

Returns `FE_Success` if the operation is successful, else sets FA_errno to `FE_CMSFailedCheckin`

Example

The following code checks in a file:

```
. . .
FA_errno=0;
if (ConnId > 0 && objectId)
{
F_IdValuePairsT *tval = (F_IdValuePairsT
*)F_Alloc(sizeof(F_IdValuePairsT), NO_DSE);
tval->len = 6;
tval->val = (F_IdValuePairT *)F_Alloc(sizeof(F_IdValuePairT) *
tval->len, NO_DSE);
ASSIGN_ID_INT(0, FV_CMSCommandCheckinMakeCurrentVersionId,1);
ASSIGN_ID_INT(1, FV_CMSCommandCheckinKeepLocalCopyId, 0);
```

```
ASSIGN_ID_INT(2, FV_CMSCommandCheckinMinorVersionId, 1);
ASSIGN_ID_STR(3, FV_CMSCommandCheckinDescriptionId, "desc");
ASSIGN_ID_STR(4, FV_CMSCommandCheckinVersionLabelId, "label");
ASSIGN_ID_STR(5, FV_CMSCommandCheckinCommentId, "comment");
StatusT result=F_ApiCMSCheckin(ConnId, objectId, tval);
objectId = 0;
}
```

F_ApiCMSCancelCheckout

Cancels check out of a file from the CMS

Synopsis

```
#include fcmsapi.h
. . .
StatusT       F_ApiCMSCancelCheckout(F_ObjHandleT cmsSessionId,
                                     F_ObjHandleT objectId);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object

Returns

Returns `FE_Success` if the operation is successful, else sets FA_errno to `FE_CMSFailedCancelCheckout`

Example

The following code cancels the check out of a file:

```
. . .
FA_errno=0;
if (ConnId > 0 && objectId)
{
StatusT result=F_ApiCMSCancelCheckout(ConnId, objectId);
objectId = 0;
}
```

F_ApiCMSDelete

Deletes a file or a folder from CMS

Synopsis

```
#include fcmsapi.h
. . .
StatusT      F_ApiCMSDelete(F_ObjHandleT cmsSessionId,
                            F_ObjHandleT cmsObjectId,
                            const F_IdValuePairsT *deleteParams);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`deleteParams`	Id value pairs to specify the delete parameter.The valid Ids are: FV_CMSCommandDeleteAllVersionId FV_CMSCommandDeleteAllDependentsId

Returns

Returns `FE_Success` if the operation is successful, else sets FA_errno to `FE_CMSFailedDelete`

Example

The following code deletes a file with all its dependents:

```
. . .
FA_errno=0;
ConStringT urlPathOrObjId =
F_ApiCopyString((ConStringT)"/libs/source.fm");
F_ObjHandleT cmsObjId = F_ApiGetCMSObjectFromPath(ConnId,
urlPathOrObjId);
if (cmsObjId)
{
F_IdValuePairsT *tval = (F_IdValuePairsT
*)F_Alloc(sizeof(F_IdValuePairsT), NO_DSE);
tval->len = 2;
tval->val = (F_IdValuePairT *)F_Alloc(sizeof(F_IdValuePairT) *
tval->len, NO_DSE);
ASSIGN_ID_INT(0,FV_CMSCommandDeleteAllDependentsId,1);
```

```
ASSIGN_ID_INT(1,FV_CMSCommandDeleteAllVersionId, 0);
StatusT result=F_ApiCMSDelete(ConnId, cmsObjId, tval);
}
```

F_ApiCMSOpenFile

Opens a file or a book from CMS in FrameMaker

Synopsis

```
#include fcmsapi.h
. . .
F_ObjHandleT F_ApiCMSOpenFile(F_ObjHandleT cmsSessionId,
                              F_ObjHandleT cmsObjectId,
                              const F_IdValuePairsT *openParams);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`openParams`	Id value pairs to specify the open parameter.The valid Ids are: FV_CMSCommandOpenReadOnlyId FV_CMSCommandSilentOpenId

Returns

Returns the handle of the file or book if the operation is successful. Else sets FA_errno to `FE_CMSFailedOpenFile`

Example

The following code opens a file in open read only mode:

```
. . .
FA_errno=0;
ConStringT urlPathOrObjId =
F_ApiCopyString((ConStringT)"/libs/source.fm");
F_ObjHandleT cmsObjId = F_ApiGetCMSObjectFromPath(ConnId,
urlPathOrObjId);
if (cmsObjId)
{
```

```
F_ObjHandleT docId;
F_IdValuePairsT *tval = (F_IdValuePairsT
*)F_Alloc(sizeof(F_IdValuePairsT), NO_DSE);
tval->len = 2;
tval->val = (F_IdValuePairT *)F_Alloc(sizeof(F_IdValuePairT) *
tval->len, NO_DSE);
ASSIGN_ID_INT(0,FV_CMSCommandOpenReadOnlyId,1);
ASSIGN_ID_INT(1,FV_CMSCommandSilentOpenId, 0);
docId = F_ApiCMSOpenFile(ConnId, cmsObjId , tval);
}
```

F_ApiCMSUploadObject

Uploads a file or a folder into the CMS

Synopsis

```
#include fcmsapi.h
. . .
StatusT       F_ApiCMSUploadObject(F_ObjHandleT cmsSessionId,
                                   F_ObjHandleT cmsObjectId,
                                   ConStringT localFilePath);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object
`localFilePath`	The full pathname of the file or folder to upload

Returns

Returns `FE_Success` if the operation is successful. Else sets FA_errno to `FE_CMSFailedUploadObject`

Example

The following code uploads a file into CMS

```
. . .
FA_errno=0;
ConStringT urlPathOrObjId =
F_ApiCopyString((ConStringT)"/libs/");
```

```
ConStringT uploadFilePath =
F_ApiCopyString((ConStringT)"C:\\Documents and
Settings\\aggrawal\\Desktop\\Generic CMS\\Book\\source.fm");

F_ObjHandleT cmsObjId = F_ApiGetCMSObjectFromPath(ConnId,
urlPathOrObjId);

if (cmsObjId)

StatusT result=F_ApiCMSUploadObject(ConnId,cmsObjId,
uploadFilePath);
```

F_ApiCMSDownloadObject

Downloads a file from the CMS

Synopsis

```
#include fcmsapi.h
. . .
StringT       F_ApiCMSDownloadObject(F_ObjHandleT cmsSessionId,
                                     F_ObjHandleT cmsObjectId);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`cmsObjectId`	The ID of the CMS Object

Returns

Returns the local file path of the downloaded file if the operation is successful. Else sets FA_errno to `FE_CMSFailedDownloadObject`

Example

The following code download a file from CMS:

```
. . .

FA_errno=0;

ConStringT urlPathOrObjId =
F_ApiCopyString((ConStringT)"/libs/book.book");

F_ObjHandleT cmsObjId = F_ApiGetCMSObjectFromPath(ConnId,
urlPathOrObjId);

if (cmsObjId)

StringT path=F_ApiCMSDownloadObject(ConnId,cmsObjId);
```

F_ApiGetCMSObjectFromPath

Gets CMS object from a URL path

Synopsis

```
#include fcmsapi.h
. . .
F_ObjHandleTF_ApiGetCMSObjectFromPath(F_ObjHandleT cmsSessionId,
ConStringT urlPath);
```

Arguments

`cmsSessionId`	The ID of the CMS Session
`urlPath`	The url pathname of the file or folder

Returns

Returns the handle of a CMS object if the operation is successful. Else sets FA_errno to `FE_CMSFailedGetItemFrompath`

Example

The following code returned a CMS object from a URL path:

```
. . .
FA_errno=0;
ConStringT urlPathOrObjId =
F_ApiCopyString((ConStringT)"/libs/book.book");
F_ObjHandleT cmsObjId = F_ApiGetCMSObjectFromPath(ConnId,
urlPathOrObjId);
```